

High Performance Iterative X-Ray CT
with Application in 3-D Mammography
and Interventional C-arm Imaging
Systems

Hochperformante Iterative
Computertomographie mit Anwendung
in der 3-D Mammographie und C-arm
CT Bildgebung

Der Technischen Fakultät
der Friedrich-Alexander-Universität
Erlangen-Nürnberg

zur

Erlangung des Doktorgrades Dr.-Ing.

vorgelegt von

Benjamin Keck

aus

Nürnberg

Als Dissertation genehmigt
von der Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der mündlichen Prüfung:	14.10.2014
Vorsitzende des Promotionsorgans:	Prof. Dr.-Ing. habil. M. Merklein
Gutachter:	Prof. Dr.-Ing. Joachim Hornegger Prof. Dr.-Ing. Marc Stamminger

Abstract

Medical image reconstruction is a key component for a broad range of medical imaging technologies. For classical Computed Tomography systems the amount of measured signals per second increased exponentially over the last four decades, whereas the computational complexity of the majority of utilized algorithms has not changed significantly.

A major interest and challenge is to provide optimal image quality at the fewest patient dose possible. One solution and active research field towards solving that problem, are iterative reconstruction methods. Their complexity is a multiple compared to the classical analytical methods which were used in nearly all commercially available systems. In this thesis the application of graphics cards in the field of iterative medical image reconstruction is investigated. The major contributions are the demonstrated fast implementations for off-the-shelf hardware as well as the motivation of graphics card usage in upcoming generations of medical systems. The first realization describes the implementation of a commonly used analytical cone-beam reconstruction method for C-arm CT, before covering iterative reconstruction methods. Both analytical as well as iterative reconstruction methods share the compute-intensive back-projection step. In addition iterative reconstruction methods require a forward-projection step with similarly high computational cost. The introduced Compute Unified Device Architecture (CUDA) builds the basis for the presented GPU implementation of both steps. Different realization schemes are presented by combining both steps and applying minor modifications. The implementations of the SART, SIRT as well as OS-SIRT illustrate the realization of algebraic reconstruction methods. Further, a realization for the more advanced statistical reconstruction methods is described, introducing a GPU accelerated implementation of a maximum likelihood reconstruction using a concave objective function.

The achieved reconstruction performance is based on different detailed optimizations and exploitation of various technical features. In addition the performance results are evaluated for different hardware platforms – like the CPU – and for the proposed algorithms. The results implicate that for all presented reconstruction methods a significant speedup compared to a CPU realization is achieved. In example, we achieve at least a speedup factor of 10 for the presented OS-SIRT comparing a NVIDIA QuadroFX 5600 graphics card with a workstation equipped with two Intel Xeon Quad-Core E5410 processors. This is additionally supported by the comparison of the presented implementations to the CUDA alternative OpenCL underpinning the performance lead of GPUs using CUDA.

A further contribution of this thesis is the exemplary clinical application of the proposed algorithms to two different modalities: C-arm CT and 3-D mammography. These applications demonstrate the potential and importance of GPU accelerated iterative medical image reconstruction. This thesis is concluded with a summary and an outlook on the future of GPU accelerated medical imaging processing.

Kurzfassung

Eine wichtige Kernkomponente der medizinischen Bildgebung bildet die medizinische Bildrekonstruktion. Betrachtet man die Anzahl der gemessenen Signale pro Sekunde für die klassische Computertomographie in den letzten vier Jahrzehnten, so lässt sich ein exponentielles Wachstum feststellen. Im Vergleich dazu veränderte sich die Berechnungskomplexität der verwendeten Algorithmen nicht merklich.

Ein wichtiger Aspekt für die Computertomographie ist die Reduktion der applizierten Patientendosis auf ein Minimum unter der Voraussetzung weiterhin eine optimale Bildqualität zu erhalten. Ein möglicher Lösungsansatz hierfür und Gegenstand aktueller Forschung sind iterative Rekonstruktionstechniken. Die Berechnungskomplexität dieser ist dabei allerdings ein Vielfaches gegenüber der überwiegend kommerziell verwendeten analytischen Rekonstruktionstechniken.

Ziel dieser Arbeit ist es, den möglichen Einsatz von Grafikkarten zur beschleunigten Berechnung iterativer Rekonstruktionstechniken zu untersuchen. Der Nachweis und die Erläuterung zu den erzielten performanten Realisierungen mit Hilfe dieser Technologie zählt zu den wichtigsten Erkenntnissen. Des Weiteren kann der Einsatz dieser breit verfügbaren Technologie in zukünftigen medizinischen Bildgebungssystemen nahe gelegt werden. Zusätzlich zu den Erläuterungen bezüglich der iterativen Rekonstruktionstechniken wird zunächst die Realisierung einer weit verbreitenden analytischen Rekonstruktionstechnik für die C-arm CT Bildgebung beschrieben. Ein Bestandteil dieser Rekonstruktionstechnik, die Rückprojektion, ist eine berechnungsintensive Kernkomponente die sowohl bei analytischen als auch bei iterativen Rekonstruktionstechniken benötigt wird. Im Unterschied zu den analytischen Methoden benötigen iterative Rekonstruktionstechniken zusätzlich eine weitere Kernkomponente, die (Vorwärts-)Projektion. Diese stellt dabei einen ähnlich hohen Berechnungsaufwand dar, wie die Rückprojektion. Die Basis für die Implementierungen beider Kernkomponenten bildet dabei die erläuterte parallele Programmierertechnik für Grafikkarten namens CUDA. Durch geringe Modifikationen und geschickte Kombination beider Kernkomponenten werden verschiedene Realisierungen erzielt. Die Implementierung der Rekonstruktionstechniken SART, SIRT und OS-SIRT stellt die Gruppe der algebraischen Rekonstruktionsalgorithmen dar. Des Weiteren wird ein Implementierungsbeispiel für die Gruppe der statistischen Rekonstruktionstechniken anhand einer Maximum Likelihood Rekonstruktion auf Basis einer konkaven Zielfunktion erläutert.

Die erzielten Rekonstruktionsgeschwindigkeiten basieren auf verschiedenen dargelegten Optimierungstechniken und der Verwendung neuer technischer Funktionen. Zusätzlich werden die Rekonstruktionsgeschwindigkeiten zwischen verschiedenen Hardware-Plattformen und den verschiedenen Algorithmen verglichen. Die Ergebnisse implizieren, dass für alle vorgestellten Rekonstruktionstechniken mittels Grafikkarten ein signifikanter Geschwindigkeitszuwachs im Vergleich zu einer CPU erzielt werden kann. Zum Beispiel ist die Implementierung der OS-SIRT auf einer NVIDIA QuadroFX 5600 mindestens Faktor 10 schneller als die CPU Implementierung auf einer Workstation mit zwei Intel Xeon Quad-Core E5410 Prozessoren. Dies wird durch den Vergleich der erläuterten Implementierungen zur CUDA Alternative OpenCL zusätzlich bekräftigt, der die Geschwindigkeitsvorteile von Grafikkarten unter der Verwendung von CUDA untermauert.

Ein weiterer Beitrag dieser Arbeit ist die exemplarische Klinische Anwendung der vorgestellten Algorithmen für zwei unterschiedliche Modalitäten: C-arm CT und 3-D Mammographie. Diese Anwendungen zeigen das Potential und die Bedeutung der GPU beschleunigten iterativen Medizinischen Bildrekonstruktion auf. Abschliessend wird die Arbeit zusammengefasst und ein Ausblick auf die zukünftige Ausrichtung der Medizinischen Bildverarbeitung gegeben.

Acknowledgment

I would like to take this opportunity to thank my advisor Prof. Dr. Joachim Hornegger (LME, University of Erlangen) for so many things. First for encouraging, motivating and fascinating me for the research field of medical image processing. For giving me your confidence and support and for being a good friend over the last decade. I am grateful for the numerous discussions and your always open door. I also wish to thank Prof. Dr. Marc Stamminger (LGDV, University of Erlangen) for reviewing my thesis.

I would like to give very special thanks to Dr. Markus Kowarschik (Siemens Healthcare) for his influence and guidance on this research, for many early morning meetings, motivation and his counseling over the years. Additionally, I would like to thank my former colleagues at Siemens Healthcare CP CV ME for their support and welcoming me in the group and the financial support. The same is true and I am thankful to the collaborating experts from Siemens Healthcare: Dr. Anna Jerebko, Dr. Thomas Mertelmeier and Dr. Günter Lauritsch. Thank you for your support.

This work was invaluable supported by Dr. Holger Scherl (Siemens Healthcare), Dr. Holger Kunze (Siemens Healthcare), Ingo Schasiepen (Siemens Healthcare) and Hannes Hofmann (LME). In addition to all the technical discussions, thank you, Holger S., for persuading me in San Diego and introducing me to CUDA. Thank you, Holger K., for your time, discussions and sharing your expertise with me in the field of iterative reconstruction. Without my LME roommate Hannes this work wouldn't be the same: Thank you for the long discussions, your friendly ear and releasing so much energy and motivation for this field of research.

Furthermore, I would like to thank all my former colleagues at the LME for being on the diagonal of the matrix and the great years at the lab. Especially, I appreciate the friendly and productive work with Dr. Christopher Rohkohl, Dr. Andreas Maier, Dr. Michael Balda and Dr. Marcus Prümmer. I would like to express my gratitude to all the students with whom I had the pleasure to collaborate over the time and their influence on this research: Andreas Weinlich, Elmar Bührle and Christian Siegl.

I also owe thanks to Dr. Karl Schwarz for pushing me over the last 3 years to finish the writing of this thesis. Last but not least, I would like to thank my family and my friends for their patience and dispensing me so often over the last years.

Erlangen, 2013-10-27

Benjamin Keck

Contents

Chapter 1	Introduction	1
1.1	History of CT and its Application and Complexity	1
1.1.1	Development of Computed Tomography	2
1.1.2	Computed Tomography Applications.	4
1.1.3	Increasing Complexity in CT.	8
1.2	Original Work and Outline of this Thesis.	11
1.2.1	State-of-the-Art	11
1.2.2	Contribution of this Work.	14
1.2.3	Outline of this Thesis.	16
Chapter 2	GPU Programming / CUDA Acceleration	19
2.1	Parallel Evolution	19
2.1.1	Immense Compute Power.	20
2.1.2	Extreme Memory Bandwidth	21
2.1.3	CPU versus GPU	21
2.2	GPGPU History	23
2.2.1	Programmable Graphics Pipeline	24
2.2.2	Load Imbalance.	25
2.3	Compute Unified Device Architecture	26
2.3.1	Programming Model	27
2.3.2	Execution Model	29
2.3.3	Memory Model	31
2.3.4	Utilized CUDA GPUs	35
2.3.5	Optimizations	36
2.4	Future Parallel Programming	38
2.4.1	Open Computing Language	38
2.4.2	DirectCompute	40
2.4.3	CUDA x86.	40
2.4.4	Fermi	40
2.5	Conclusion	42
Chapter 3	Medical Image Reconstruction Algorithms	43
3.1	Reconstruction in General	43
3.1.1	Geometry and Acquisition	43
3.1.2	Reconstruction Methods Overview	50
3.2	Analytical Reconstruction Methods.	54
3.2.1	Fourier Slice Theorem	54
3.2.2	Filtered Back-Projection	55
3.2.3	The FDK method	56

3.3 Algebraic Reconstruction Methods	57
3.3.1 Depiction of the System Matrix Elements.	57
3.3.2 Algebraic Reconstruction Technique	58
3.3.3 Simultaneous Algebraic Reconstruction Technique	59
3.3.4 Simultaneous Iterative Reconstruction Technique.	60
3.3.5 Ordered Subsets for SIRT	61
3.4 Statistical Reconstruction Methods	62
3.4.1 Maximum Likelihood using Expectation Maximization	64
3.4.2 Maximum Likelihood using Gradient-based Optimization	65
3.4.3 Maximum Likelihood Convex Algorithm	65
3.4.4 Regularization - Constraints Incorporating Priors	70
3.5 Convergence and Complexity.	71
3.6 Conclusion	72
Chapter 4 High Performance Medical Image Reconstruction	73
4.1 High Performance CUDA Implementations	74
4.1.1 Fast Fourier Transformation	74
4.1.2 Pre- and Post-Processing	75
4.1.3 Back-Projection	76
4.1.4 Forward-Projection	83
4.2 High Performance Medical Image Reconstruction Approaches	97
4.2.1 Filtered Back-Projection and the FDK method	97
4.2.2 Simultaneous Algebraic Reconstruction and its Variations.	99
4.2.3 Maximum Likelihood Reconstruction	106
4.3 OpenCL Comparison	111
4.4 Conclusion	112
Chapter 5 Clinical Applications	113
5.1 Interventional C-arm CT	113
5.2 3-D Mammography.	118
5.3 Conclusion	124
Chapter 6 Summary and Outlook	125
6.1 Summary	125
6.2 Outlook	127
Chapter A Datasets	129
A.1 DataSet A – Human Head	129
A.2 DataSet B – Human Hip	129
A.3 DataSet C – Simulated Head	129
A.4 DataSet D – Catphan CTP 528	130
A.5 DataSet E – Simulated Head	130
A.6 DataSet F – Female Breasts	130

Appendix B EM Reconstruction for Computed Tomography	131
B.1 Maximum Likelihood - Expectation Maximization	131
B.2 EM Algorithms for Transmission Tomography	132
List of Figures	135
List of Tables	137
Bibliography	139

Introduction

In this thesis we address specific research topics in the field of accelerating medical image processing algorithms using off-the-shelf hardware, especially modern computer graphics cards. In the last decades, development of these cards was driven dramatically by the gaming industry. But more and more researchers and manufacturers realized their potential compute power.

Already in 1998, Mueller *et al.* [Muel98b] made use of the programmable graphics pipeline for medical image reconstruction and showed a reliable speedup – more than 10 faster comparing GPUs and CPUs at this time for lower resolutions. Further on, more and more reports and experiences on graphics cards programming for medical image processing using OpenGL, DirectX, and shading languages were published [Muel99, Muel00, Xu04, Phar05, Xu05, Schi06, Muel07, Xu07a, Hill09a, Xu10]. At the end of 2006 the interest for computations on GPUs, including medical image reconstruction, was revolutionary changed by NVIDIA introducing the first unified architecture of a Graphics Processing Unit (GPU) in their GeForce 8800 GTX graphics cards.

To understand the meaning of a unified architecture as well as the major difference compared to a non-unified architecture, it is important to know how traditional computer graphics computations were processed. In the second chapter we will explain and discuss this in detail. The relevance for medical image processing can be seen in the correlation with the history of Computed Tomography (CT) and its rising application and complexity described in the following section.

1.1 History of CT and its Application and Complexity

Four decades ago, Sir Godfrey Newbold Hounsfield invented the first Computer Tomograph in the year 1969 at EMI Central Research Laboratories in Hayes, United Kingdom. Independently Allan McLeod Cormack of the Tufts University in Massachusetts had a similar idea and published a mathematical technique [Corm63] without knowing Hounsfield's work. In 1979 Hounsfield and Cormack were awarded the Nobel Prize in Medicine for their invention of Computed Tomography [Fill10].

Since then Computed Tomography (CT) rose widely in the world not only in the field of applications, but also in its technical complexity. Specifically the amount of measured signals (readings) per time increased exponentially. The computation of transaxial slices from these measured signals is one of the main research foci of (medical) reconstruction algorithms.

In the following we describe the development of Computed Tomography and the different generations of CT systems. Afterwards the wide range of Computed Tomography applications is presented before completing the CT introduction with the increase in complexity illustrated by classical CT.

1.1.1 Development of Computed Tomography

The original prototype by Hounsfield in 1971 acquired 160 parallel readings through 180 angles, each 1° apart. Each scan lasted a little over 5 minutes. It took 2.5 hours at the EMI computer center to process the image from the first scan by Algebraic Reconstruction Technique (ART) [Doss 00]. The scanner had a single photo-multiplier detector and operated on a translate and rotate principle. This principle – shown in Figure 1.1 – is described in literature as 1st generation CT.

During the development of the first CT, Hounsfield's idea and concept did not find acceptance at the National Neurological Hospital at Queen's Square in London [Fill 10] where he first presented the idea. The statement was made by the chief of neuroradiology that with the current technologies like plane tomography and angiography, there was no brain lesion that could not be diagnosed by imaging already [Fill 10]. Hounsfield went to the number two neurological hospital in London and managed to solicit their chief of neuroradiology – Jamie Ambrose at the Atkinson Morley's Hospital in Wimbledon. While the entire staff members of the hospital were sworn to secrecy, the machine was built along a plan for commercial production. After the public announcement the first professional assessment by the National Neurological Hospital was clearly disproved and a wide range of researchers, physicians and also manufacturers picked up the idea and saw its opportunities. For example, the medical equipment manufacturer and one of the leading companies in healthcare technology – Siemens – established its own CT development within its basic research department in 1972 [Wiec 04].

It is not surprising, that after the first CT scanner which was specifically used for Brain CT, the technology evolved. While the EMI head scanner went into clinical service in 1973, competitors began to introduce faster scanners. To reduce the acquisition time, the next generation scanners measured multiple readings instead of a single one. The fan-beam geometry utilizing a detector array is the distinguishing feature of the 2nd generation CT [Doss 00]. These systems were still using a translate-rotate scanning motion, illustrated in Figure 1.1, but were achieving a better performance by measuring a fan-beam concurrently. The first introduced scanners of this generation used three detector elements per fan. Soon, twelve and more detector elements [Boyd 90] were manufactured. In example this enabled a fan angle of 10° degree for about 30 detector elements [Doss 00].

In 1975 Hounsfield kept improving CT scanners by introducing the first whole-body CT. The EMI-5005 was brought to market in 1976 achieving an astonishing acquisition time of 18 seconds per scan using a 20 elements detector [Chik 78]. The acquisition time was reduced below 20 seconds which was considered as a breath-holding interval. Therefore a scan through the thorax, with almost no movement of the thorax, was possible and just one application of whole-body CT.

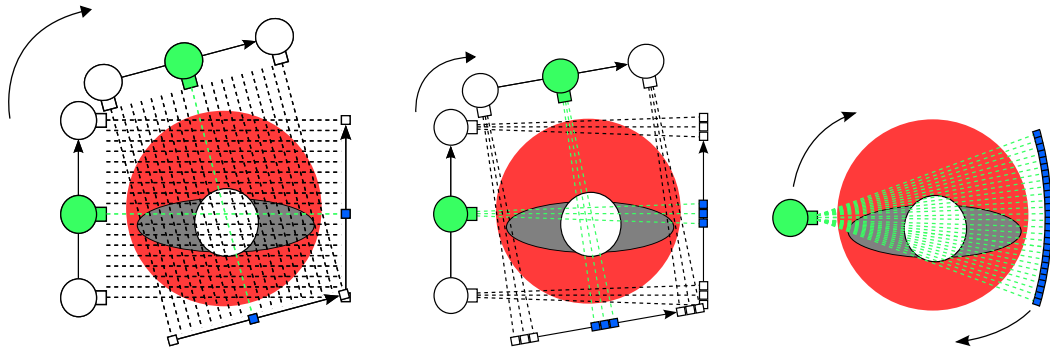


Figure 1.1: The figure illustrates from left to right the different generations of CT systems, starting with the 1st generation CT acquiring each measurement by step and shoot. The 2nd generation CT then acquired already multiple measurements by step-and-shoot. Finally the 3rd generation CT can acquire a significant detector array and enables a continuous rotation.

During the period 1974-1977 several companies were developing a new class of scanner that would require only rotary motion using a broader fan-beam. In 1977 Siemens introduced this new class of scanner systems that integrated the radiation source and a detector system into what became known as a “gantry” [Schm04]. The mechanism rotates around the patient and allows whole-body Computed Tomography examinations. These systems are also known in literature as 3rd generation CT systems (shown in Figure 1.1). The first systems were able to do a 360° degree rotation. Since the X-ray source and the detector elements were connected via cables, the gantry had to stop after one rotation and continued rotating in the opposite direction. A crucial improvement came with the development of the slip ring technology in 1987. A bidirectional connection to the components via these slip rings was enabled while the gantry could rotate continuously. This technology built the basis of further milestones in the history of Computed Tomography.

In 1989 Siemens introduced the first helical CT scanner. Kalender was credited for the invention and development of helical scan Computed Tomography. The advantages of such systems were soon explored. For example, a helical CT was able to scan a patient’s complete lung – instead of just a part – during a single breath-hold [Kale90].

So far, the X-ray emitter and the detector elements of the CT system were moving around the object on a circular trajectory. Due to the continuous movement of the gantry, it was possible to introduce a continuous movement of the scanned object through the CT system. The rotating acquisition around the moving object ends up in a helical trajectory. The advantages of a helical acquisition is a fast scanning of the patient – possible from tip to toe – with less artifacts from patient movement due to the continuous and non-sequential movement of the table.

In literature a 4th generation of CT systems is also described [Buzu08]. This generation of CT systems provides also a rotating X-ray source, but instead of a rotating detector utilizes a closed stationary detector ring. Due to the fact that such systems were never introduced to the market, we do not address them in this thesis and focus on further milestones of 3rd generation CT systems.

To improve scanning speed and reduce slice thickness, manufacturers were still improving their CT systems. In 1991, a multi-slice CT was first demonstrated [Buzu 08] and 7 years later, in 1998, Siemens introduced their first multi-slice CT scanner to the market. The SOMATOM Volume Zoom was capable of acquiring 4 slices per rotation.

To be able to acquire 4 slices per rotation simultaneously, the manufacturers added multiple detector arrays. The acquisition geometry changed from a fan-beam to a cone-beam. Over the years the amount of detector elements along the rotation axes were steadily increased by the manufactures of CT systems. The maximal number so far provides Toshiba's Aquilion ONE utilizing as much as 320 detector arrays with 896 detector elements per row [Cent 09b].

One recent milestone is the development of multi-source CT. These types of CT systems are equipped with two multi-slice CTs, wherein the two source/detector combinations simultaneously collect data. Both X-ray tubes are oriented in about 90° degrees to each other. In 2006 Siemens introduced their first dual source CT system. Flohr et al. [Floh 06] demonstrated different kinds of improvements in CT using this system. A dual source system also enables an acquisition using tubes at two different energy levels. This results in a dual energy reconstruction, introduced by Alvarez et al. [Alva 76] already in 1976.



Figure 1.2: Two examples of CT systems and their application. On the left a Siemens SOMATOM Definition DS illustrating a beating heart acquisition. On the right side the latest Siemens SOMATOM Definition Flash is used for a brain perfusion measurement. Images by courtesy of Siemens AG

1.1.2 Computed Tomography Applications

After discussing important steps in the development of Computed Tomography, the range of this technology is summarized to describe the wide field of applications where our research results can be applied. There is no question that the 3rd generation CT device is one of the most distributed diagnostic medical device in the world. By June 1975 the UK-based music, electronics, and leisure company – EMI Limited



Figure 1.3: On the left a combination of a SPECT and CT system, the Siemens Symbia, is shown. The Siemens Biograph mCT as an example for the combination of PET and CT is depicted on the right side. Images by courtesy of Siemens AG

– still led the CT market and had shipped 122 [Bart83] of head and whole-body CT-systems in total, whereas Siemens had sold only two Siretom CT systems in the year of 1974 [Wiec04]. In 1990 more than 15000 CT scanners [Boyd90] were installed throughout the world and in 2004 Siemens produced as much as 2000 of their CT systems [Wiec04]. Today the innovation leader – Siemens – produces about 2000 systems per fiscal year. The broad application range of CT systems is described in literature [Doss00, Kram07, Buzu08]. Thus, we will shortly give an overview of important applications.

One of the early applications of CT systems was the visualization of the brain to detect aneurisms in order to diagnose or prevent strokes by further treatment. Today, classic CT systems are able to automatically identify nodules inside a patients' lung in order to indicate possible tumors [Wiec04]. They enable virtual flights through the human colon – out of CT acquired data – for the physician to detect polyps. After the surgical elimination of these colon polyps an outbreak of colon cancer can be prevented with high probability [Wiec04].

A wide range of applications in classical CT is based on perfusion measurements. Common types are brain perfusion – e.g., to measure the cerebral blood flow and the cerebral blood volume – and perfusion inside the liver or tumors. Mostly, application specific contrast agent has to be used in combination with fast CT scanners to actually measure the perfusion. Systems are also specialized for Coronary CT Angiography (CTA) to non-invasively display the coronary vessels for calcifications and stenoses detection [Wiec04]. With dual-source CT systems, it is possible to actually visualize the beating heart in three dimensions [Hsia10].

An important consideration is the usage of CT systems as a planning tool. All kind of surgeries as well as radiotherapies are planned using patient information out of a classical CT system [Buzu08]. Ultimately, this broad range of applications and the terrific acquisition speed of current CT systems accomplishes the fast diagnoses of a trauma patients' whole body as well as improves the treatment of the patient in the first *golden hour* after the accident [Nico08].

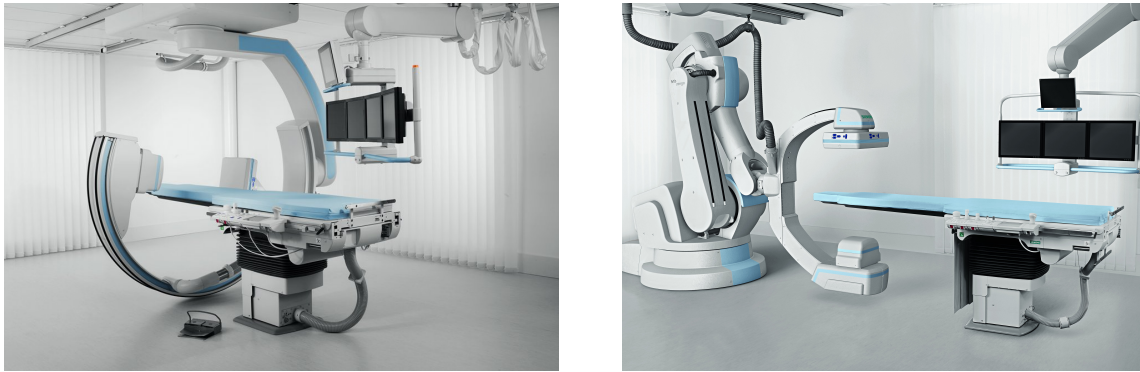


Figure 1.4: The Siemens Axiom Artis – mounted on the ceiling – in the left image and the Siemens Artis ZeeGo – integrating industrial robot technology – on the right side demonstrate the flexible movability of state-of-the-art C-arm CT systems. Images by courtesy of Siemens AG

While the history of Computed Tomography is pointed out by classical CT systems, the technology itself over the time made its way into specialized diagnostic machines such as Micro CT, Dental CT or Breast CT systems. Computed Tomography is also used for quality assurance and testing of materials and products – named Non-Destructive Testing (NDT).

In the last decade classical CT systems were more and more combined with different diagnostic technologies. Not only in the way of merging differently acquired datasets, but also by combining these technologies in a single device. Examples for such systems – so called “hybrids” – are the combinations of Single Photon Emission Computed Tomography and CT (SPECT/CT) or Positron Emission Tomography and CT (PET/CT). Examples for both systems are shown in Figure 1.3.

We demonstrate the application of our research results on two specific systems. These are pointed out next, before going on with the increase in complexity of Computed Tomography justified by classical CT.

The first specific system and application concerns Computed Tomography in the field of interventional procedures. The main interest in Angiography is the visualization of blood vessels and organs of the human body. Therefore, Angiographic C-arm systems provide high-resolution 2-D X-ray images in real-time for surgery guidance.

The flexible movement of C-arm systems also allows the usage for tomographic 3-D imaging – so called C-arm CT. This capability was examined and published more than 20 years after the invention of Computed Tomography [Fahr 97]. Since then quality, usability and applications were continuously improved [Stro 09]. The basic principle is similar to multi-slice CT. In this case a C-arm instead of a gantry rotates around the patient and acquires a set of 2-D X-ray projections. Similar to the first generation CT these systems perform not a full 360° rotation. Instead an adequate half rotation is induced. For these systems therefore it is $180^\circ + \text{fan-angle}$. Figure 1.4 shows two examples of state-of-the-art C-arm systems.



Figure 1.5: Siemens ARCADIS Orbic a mobile C-arm CT system. Image by courtesy of Siemens AG

Since these 3-D imaging applications are generally used during intervention, reconstruction times and delays are critical. Details about the application of C-arm CT can be found in Strobel et al. [Stro09]. C-arm CT systems are geometrically very flexible, but require also remarkable amount of space. Therefore smaller and moveable mobile versions were introduced. These smaller system are also capable of Computed Tomography. An example for such a device is illustrated in Figure 1.5.

The second application covered in this thesis is 3-D mammography. In the western world breast cancer is the most common form of cancer for females [RKI 12]. Up to 8 – 10% of all females develop breast cancer in their lifetime [Siem04]. In Germany these are around 46000 females p.a., around 17000 of whom are younger than 60 years old. To prevent breast cancer screening examinations were established. Mammography is commonly used as a screening method. The images are then inspected by two physician on an average of five minutes [Enzm01] for interpretation of these images. While 3-D mammography provides many advantages compared to a 2-D mammography, it is also more compute intensive and especially more time consuming. This is critical in the case of screening methods, where throughput is an important economical aspect.

Speaking of 3-D in this case might be exaggerating due to the fact that methods of digital tomosynthesis are used. Although there are some similarities to CT, it is a separate technique. Tomosynthesis allows to generate an arbitrary number of in-focus planes retrospectively from a sequence of projection radiographs that are acquired during a single motion of the X-ray source. Dobbins et al. summarize the history and advantages of tomosynthesis as well as its clinical potential in [Dobb03]. To name important facts, Tomosynthesis is already known since the 1930s. Even decades before the development of CT, tomographic imaging provided for physicians three important advantages over conventional projection radiography:

1. Due to varying shifts in acquisition, planes at different depths can be rendered in focus, such that it permits a depth localization of structures



Figure 1.6: The Siemens MAMMOMAT Inspiration. In the left image detector, compression plate and X-ray source illustrate the acquisition system. The right image exemplifies an examination. Images by courtesy of Siemens AG

2. Improved conspicuity of structures by decreasing the influence of interfering attenuation from overlaying anatomy
3. Contrast of local structures is improved by restricting the overall image dynamic range to that of a single slice

In digital breast tomosynthesis a series of consecutive two dimensional projections of a compressed breast at different angles are acquired. To be more specific, such acquisitions are performed over an angular sweep of thirty to fifty degrees. Within this sweep eleven to thirty projection-views are measured. This incomplete set of data can be digitally processed to yield images similar to conventional tomography, but is unable to offer the extremely narrow slice widths that CT provides. Figure 1.6 gives an impression of a state-of-the-art 3-D mammography system and the acquisition procedure.

1.1.3 Increasing Complexity in CT

The significance of High Performance Computing (HPC) in Computed Tomography is argued by the increasing complexity in CT over the last decades. On one hand, the complexity is related to the complexity of the applied reconstruction algorithm. We will discuss the complexity of the utilized reconstruction algorithms in Chapter 3. On the other hand, the complexity depends on the amount of data used for the reconstruction of transaxial slices. In case of time critical imaging the amount of data per time is relevant.

The original EMI head scanner used an 80^2 image matrix. Most scanners today use 512^2 up to 1024^2 image matrices. Boyd et al. [Boyd 90] state a rule of thumb: for CT fan-beam scanners using a image resolution of $n \times n$ the number of angular

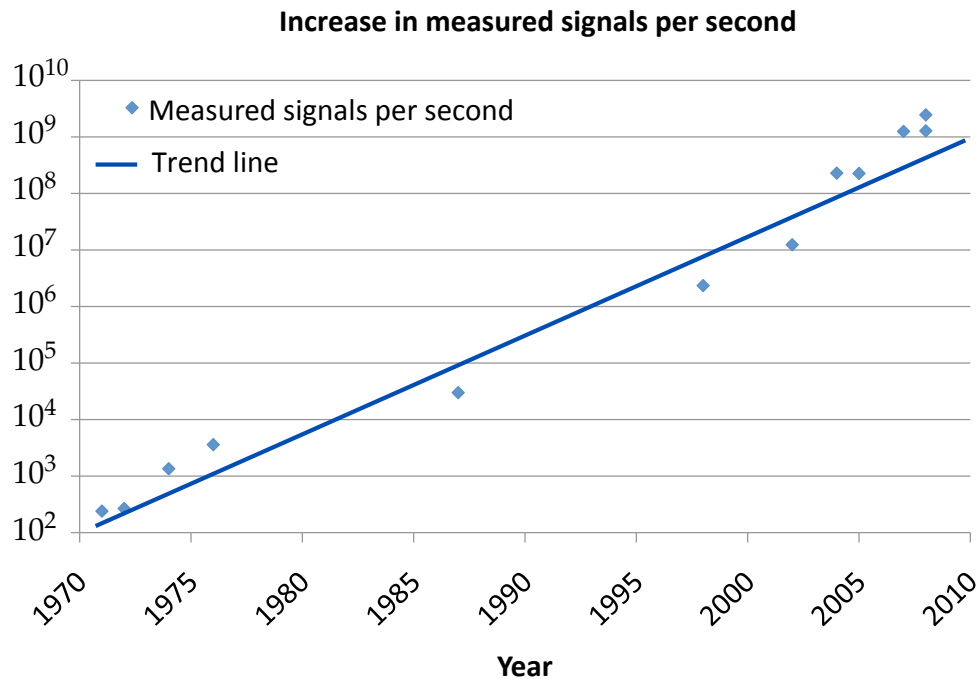


Figure 1.7: An exponential increase of measured signals per second for classic CT system over the last four decades is given by measurement points for several CT systems. Please note the logarithmic scale of the plot.

samples must be $2n$ and the number of samples in a projection should be $1.4n$. As n becomes larger resolution improves, but the total number of attenuation measurements increases with $O(n^2)$. For details on the sampling theorem in CT we refer to the literature [Natt 86].

Over the last four decades CT manufacturers increased the resolution, but also accomplished an incredible decrease in acquisition time. Boyd et al. already presented the CT scan speed development and its historical progression from 1972 till 1990 in [Boyd 90]. The immense increase in measured signals per second – depicted in Figure 1.7 – depends on the following improvements. First the rotation speed was increased. The biggest improvement was the introduction of the gantry facilitating a continuous rotation. Thereafter, the rotation time – limited by physics and gravitation – was piecewise accelerated by new technologies. The latest high-end CT systems can perform up to 3.5 rotations per second [Cent 09b]. All components mounted on the inside of the gantry have to withstand extreme gravitational forces – more than 30g for a rotation speed of 0.28s. Important to note is that such speed improvements were only possible due to better and faster X-ray tubes as well as the development of new detectors. E.g., Siemens developed the STRATON X-ray tube as well as a detector made of Ultra Fast Ceramic (UFC).

In the second generation of CT systems the number of detector elements/channels per detector row evolved up to 12 – 60 [Chik 78]. Third generation CT systems introduced significantly larger fan-angles and therefore a significant higher amount of detector elements. The Siemens SOMATOM Plus from 1987 used 416 channels, while the Philips Brilliance iCT from 2005 uses as much as 672 channels. The age of

multi-slice CT scanners multiplied the amount of detector elements by introducing several detector arrays along the rotation axis.

Next specific technical inventions like the quarter detector shift, flying focus technology or z-sharp technology are utilized to improve geometrical information as well as increasing the amount of readings using the same detector arrays [Cent 09a]. At last dual-source CT systems – motivated by even faster acquisition time and dual-energy acquisitions – basically consist of two CT systems and were again raising the peak of measurements per second for CT scanners.

All these improvements over the last 40 years raised the growth of readings per time in Computed Tomography. In Figure 1.7 we depict this exponential increase for the last four decades using the amount of measured signals per second for several classic CT systems. In 2007 Polacin *et al.* analogously showed that the increasing amount of slices in combination with shorter scan time leads to very high input data rates, which has to be handled (see [Pola 07] Fig.2). Technical details about the classic CT scanning devices from the last decade can be found in [Cent 09a, Cent 09c, Cent 09b].

The exponential increase of measurement data is an important motivation for this work. Therefore, the data rates for the two focussed modalities are detailed as well. In 3-D mammography 25 projections – each up to 2816×3584 readings – are acquired in about 5 seconds.

For interventional C-arm CT digital flat-panel detectors with a resolution up to 2480×1920 or 2048^2 are used. Typically a 2×2 binning of the acquired data is directly applied in the detector such that the provided data provides a projection resolution of up to 1240×960 or 1024^2 pixels.

The C-arm of the Siemens Axiom Artis can rotate 220° in about 3 seconds. The successor – Siemens Artis ZeeGo – theoretical can perform the same rotation even faster, but this is currently not allowed due to security reasons. These systems currently acquire – depending on the protocol – from 130 up to 550 projections per sweep.

1.2 Original Work and Outline of this Thesis

In this thesis we present and evaluate different approaches of GPU accelerated image reconstruction algorithms. We argue application specific why the acceleration is essential for state-of-the-art and future Computed Tomography systems. An overview of state-of-the-art accelerated medical image reconstruction is given before the contribution of our work in this field of research is stated.

1.2.1 State-of-the-Art

To overcome the exponential increase – of acquired data per time and its digital processing – manufacturers focussed on developing specialized hardware in the last decade. The developed components are installed in order to meet the requirements for the computing time of clinical systems. Therefore large parts of this research area were driven by commercial interests. Instead of computing centers or several computers most manufacturers utilized specifically developed acceleration cards [Heig 07] equipped with Field-Programmable Gate Arrays (FPGA). The Cell processor – introduced in 2005 – came also to manufacturers' and researchers' attention. First publications on accelerated reconstruction using this architecture were made by Kachelrieß *et al.* [Kach 07] and Scherl *et al.* [Sche 07c], while for commercial usage Mercury Computer Systems for example offered an acceleration card [Bock 07].

In the end of 2006 NVIDIA introduced their new graphics processing unit offering more than 300 GFlops of peak performance. By the turn of that year this was heavily recognized by both researchers and manufacturers. Hence many works on GPU accelerated reconstruction were published [Riab 07, Sche 07b, Vaz 07, Xu 07a, Xu 07b, Yang 07]. To mention the important publications we will divide them into different groups. First the computationally less compute intensive analytical reconstruction algorithms. Both other groups are on iterative reconstruction methods. The second group focus on algebraic reconstruction methods and the third group on statistical reconstruction methods. Each group will be further split by their programming language.

Analytical Reconstruction

In the year 1997 Sabine Iserhardt describes in her thesis [Iser 97] one of the early realization of an cone beam reconstruction using the hardware supported texture mapping of an SGI Indigo² High IMPACT workstation. Mueller *et al.* is clearly one of the early adopters for medical image reconstruction accelerated by graphics cards [Muel 98b]. Since 1998, he and his group are continually contributing to this field of research. Their research also includes analytical reconstruction using OpenGL and shading languages. Important to mention is Xu *et al.* with a 3-D computed tomographic reconstruction on a NVIDIA GeForce 8800 GTX graphics card [Xu 07a]. Vaz *et al.* – from Barco Medical Imaging Systems – published also an OpenGL approach with similar performance [Vaz 07] using a Barco MXRT 7200 GPU – based on AMD GPU RV600 class. Their software also offers a 2-D real-time visualization

during the reconstruction. Hillebrand *et al.* [Hill 09a, Hill 09b] published an interactive CT reconstruction using OpenGL on a NVIDIA GPU. The reconstruction is based on filtered back-projection. All pre-filtered projection images persist in the device memory of the GPU. The reconstructed slice then can be geometrically modified, while for each configuration the slice reconstruction is performed in real-time. Such they gain a highly interactive reconstruction and viewing of reconstructed 2-D slices.

One of the early approaches using the CUDA programming language was published by Riabkov *et al.* [Riab 07]. They apply a filtered back-projection reconstruction for their mobile C-arm CT and show a reasonable speedup compared to CPU solutions with roughly the same performance for the same graphics card as Xu *et al.* Excepting our research results we would like to mention two more recent CUDA-based analytical reconstruction acceleration approaches. In 2009 Bi *et al.* [Bi 09b] published results on real-time visualization of 3-D reconstruction using CUDA. Instead of a 2-D visualization, they perform a 3-D rendering of the volume during the reconstruction using the NVIDIA Tesla C1060 GPU. Last, Ino *et al.* [Ino 10] published an out-of-core cone beam reconstruction using multiple GPUs.

An early competition to NVIDIA's CUDA language is the less common ATI CAL language by AMD. Wang *et al.* [Wang 09] presented that AMD graphics cards can achieve comparable performance to NVIDIA's graphics cards. With a promising future a new programming language – called OpenCL – was already focus of research of Zhang *et al.* [Bi 09a] and Wang *et al.* [Wang 10]. Both show the possible acceleration of analytic reconstruction using OpenCL in combination with GPUs. So far the performance results are not comparable with the better performing CUDA approaches. We will mention and update this in Chapter 6 – where we discuss the future of GPU accelerated medical image processing.

Algebraic Reconstruction

The first GPU accelerated algebraic reconstruction was presented by Mueller *et al.* in 1998 [Muel 98b] utilizing the Open Graphics Library (OpenGL) and the associated shading language. His research was further refined in his group. Important to mention is the work of Xu *et al.* [Xu 10] on the efficiency of iterative ordered subset reconstruction algorithms for acceleration on GPUs as well as [Xu 09a] for GPU-accelerated SART reconstruction. They prove that due to the time costs of the update procedure in iterative reconstructions an ordered subsets approach can significantly improve the overall performance. While such an orderered subset approach provides a slower convergence, this can be neglected due to the higher performance benefit. Wei Xu published a combination of the preceding approach with a bilateral filter regularization [Xu 09b]. The filtering as well as the reconstruction are programmed with the OpenGL Shading Language (GLSL) using OpenGL.

For accelerated algebraic reconstruction using the CUDA language we want to mention the results of the following researchers. In the end of 2008, Knaup *et al.* [Knau 08] published accelerated forward- and back-projection algorithms – the basis for algebraic reconstruction – for parallel- and cone-beam tomographic imaging. Like most implementations, a voxel-based back-projection for an ideal geometry is

presented. The forward-projection is based on a 3-D parallel version of the Joseph algorithm [Jose 82]. Second is Lu *et al.* – in August 2009 – on accelerating algebraic reconstruction using CUDA-enabled GPU [Lu 09]. Analogous to the approach presented in this thesis, they use a ray-driven forward-projection step and voxel-driven back-projection. They utilize a 3-D texture array to benefit from the texture interpolation in the forward-projection step. The performance results are unfortunately only given for a small volume size of 128^3 voxels, rarely used in state-of-the-art CT. Finally the work of Pan *et al.* is considered. He published the idea [Pan 09] of Total Variation (TV) regularized iterative image reconstruction for mobile C-arm CT in February 2009, while the implementation and performance results [Pan 10] were published one year later in February 2010.

Statistical Reconstruction

The third group considers statistical reconstruction algorithms accelerated by the GPU. Most contributions focus on reconstruction algorithms with application in emission tomography where the lack of data and intense noise make statistical reconstruction algorithms indispensable. The typical volume sizes used in emission tomography are obvious smaller, such that a direct comparison to transmission tomography is very difficult. Praxt *et al.* [Prat 09] published a 3-D Ordered Subsets - Expectation Maximization (OS-EM) algorithm with application in PET based on OpenGL and shading languages. They state that even for smaller volume sizes and therefore less parallelization GPUs are becoming increasingly useful as a computing platform for medical image reconstruction.

Herraiz *et al.* introduced a similar idea in GPU acceleration of a fully 3-D iterative reconstruction software for PET using CUDA [Herr 09]. They present a loop re-ordering and optimized memory allocation for further performance improvements. Also in 2009 Nguyen *et al.* [Nguy 09] published a CUDA-based relaxed version of the OS-EM algorithm. While the application is used for image reconstruction with Compton cameras, the most time consuming operations – forward- and back-projection – were parallelized for GPU acceleration analogously. Another iterative CUDA implementation [Andr 09] – in this case for SPECT – was presented by Andreyev *et al.* The more compute intensive forward-projection based on spherically symmetric basis functions – also called “blobs” – has to be emphasized.

Finally we want to mention the work of Vintache *et al.* who published one of the few papers on GPU accelerated statistical reconstruction for Computed Tomography. This is probably due to the higher computational complexity and the immense amount of data acquired in CT. Their work on iterative reconstruction for transmission tomography on a GPU using NVIDIA’s CUDA was published in 2010. They accelerate an Ordered Subsets Convex (OSC) algorithm [Vint 10] by dividing the reconstruction in three parallel executed kernels – forward-projection, back-projection and estimated volume update operation. Unfortunately they do not detail on the exact implementation components or memory usage, making a comparison clearly impossible.

Additionally, our presented work effected recent exploratory work, e.g., research on perfusion C-arm CT [Manh 12, Manh 13] and cardiac reconstruction [Schw 13].

1.2.2 Contribution of this Work

Subsequently, an overview of the original contributions of this thesis along with the corresponding scientific publications is provided. We split the overview in four different categories.

Analytical CUDA-based reconstruction

One of the earliest publications for CUDA-based analytical reconstruction using the FDK reconstruction is [Sche 07b].

- Initial performance results of the back-projection were three times higher than the CELL performance
- Dramatic decrease in development time for CUDA and GPUs

The proposed voxel-driven back-projection represents the starting point for further reconstruction approaches like algebraic and statistical reconstruction. We show that the operator can be applied to any arbitrary acquisition trajectory using projection matrices. Further we show that the performance of our highly optimized approach is limited by the memory-bandwidth of GPUs.

Ray-driven forward-projection comparison on GPUs

A comparison of GPU implementations of a ray-driven forward-projection utilizing different programming languages and technologies was introduced in [Wein 08].

- CUDA implementations can keep up with the performance of OpenGL-based implementation utilizing 3-D textures
- Early CUDA implementations were not capable of such high performance due to missing feature support of 3-D textures

To accelerate the forward-projection – 2nd important operator for iterative reconstruction algorithms – we have introduced two CUDA implementations and compared them with a common OpenGL implementation. We can show that after the introduction of CUDA 2.0 the technology was on an equal footing with OpenGL considering a 3-D ray-driven forward-projection for arbitrary geometries.

CUDA-based algebraic reconstruction

The first performance results on 3-D CUDA-based iterative reconstruction using the Simultaneous Algebraic Reconstruction Technique (SART) were published in [Keck 09a]. The enhanced usage of new technical features for higher resolutions and reduced memory consumption were presented in [Keck 09b].

- Iterative algebraic reconstruction approaches for CUDA 1.1 and CUDA 2.0 using 2-D resp. 3-D textures
- Performance comparison and improvement due to ordered subsets (Ordered Subsets for the Simultaneous Iterative Reconstruction Technique)

- High resolution approach with slightly lower performance but less memory consumption (only 50%)

We update the performance results on our significant contribution on GPU accelerated algebraic reconstruction using CUDA. Further we present an improved version utilizing a 2-D texture from pitch linear memory for volume representation introduced in CUDA 2.3.

Statistical reconstruction utilizing CUDA-enabled GPUs

Highly efficient decomposition of the compute intensive statistical reconstruction algorithm for GPU acceleration.

- Parallel implementation of a maximum likelihood reconstruction approach with a concave objective function
- 3-D and 2-D texture usage for performance and high-resolution approaches
- Possible extension to incorporate prior usage

The GPU acceleration approach for a suitable statistical reconstruction algorithm is presented. The parallel pattern and decomposition in this case are important for performance increase while the additional memory usage of the presented approach points out possible limitations.

Our research results of CUDA-based filtered back-projection have heavily influenced product development. Since 2010 a product version of our code is included in state-of-the-art C-arm CT systems. The statistical reconstruction algorithm and its optimization can provide advanced image quality for 3-D mammography [Jere 10b]. Due to the time critical application as a screening method our GPU-acceleration approach is important for products in this field. In addition our patent application [Keck 10] on *Reduction of artifacts caused by movement of an X-ray tube in object reconstruction* needs to be named in this field. This research was extended by a comparison of CUDA to OpenCL, which was published [Sieg 11a, Sieg 11b] in 2011.

1.2.3 Outline of this Thesis

A chapter-wise overview of this thesis is provided subsequently. Additionally, a graphical structure is depicted in Figure 1.8 to provide a graphical overview of the thesis structure.

Chapter 2 – GPU Programming / CUDA Acceleration

In the second chapter, the usage of GPUs for medical image processing is motivated by picturing the performance evolution of both compute power and memory bandwidth for parallel processing. The usage of GPUs for general purpose computations and the influence on research and industry is discussed. Further an insight into GPU programming using shading languages is given. After clarifying the drawbacks of a static non-unified GPU architecture the compute unified device architecture (CUDA) and its programming concept are introduced. Two different generations of such devices and technical features are then analyzed. Specific performance optimization techniques are detailed which are used in our research approaches. An outlook for the GPU architecture successor, the Open Computing Language and its correlation to CUDA concludes this chapter.

Chapter 3 – Medical Image Reconstruction Algorithms

The used nomenclature as well as the describing geometry definition starts the third chapter. After a brief overview of reconstruction algorithms the covered reconstruction algorithms are summarized in order to prepare the prerequisites of our research. Looking at the three different groups of reconstruction algorithms the analytic reconstruction is dealt with first. Afterwards the reconstruction problem is solved by algebraic methods. Here, four reconstruction algorithms are differentiated due to the iteration scheme. Lastly, the class of statistical reconstruction algorithm is represented by the derivation of a Maximum Likelihood (ML) approach using a concave objective function. The presented reconstruction formulas then represent the basis for the presented implementations.

Chapter 4 – High Performance Medical Image Reconstruction

Prepared with an understanding of GPU acceleration as well as the utilized reconstruction algorithms, this chapter details our high performance implementations together with performance verifications. Therefore, the two major operators and their implementations are discussed after two examples for GPU acceleration are stated. The back-projection approach – as first major operator – is utilized in all three groups of reconstruction algorithms. Afterwards the forward-projection necessary for iterative algorithms is introduced as the second major operator. For this implementation the basic principle is distinguished by several kinds of GPU memory. In the second part the design of each reconstruction implementation is proposed. Using the back-projection component and FFT based filtering, the implementation concept of the FDK reconstruction is depicted. Depending on the forward-projection different kinds of algebraic reconstruction approaches are illustrated. Disadvantages

as well as advantages of each version demonstrate the limitations and their usability. Finally the combination of both major operators together with a specific update routine represents the GPU accelerated implementation of a statistical reconstruction approach.

Chapter 5 – Clinical Applications

The performance and image quality evaluation is demonstrated for two specific clinical applications. The first scenario is based on interventional C-arm CT. Due to the fact that analytical reconstruction algorithms can already be performed on-the-fly, the next class of reconstruction methods with higher complexity – named algebraic reconstruction – is proposed for possible applications. Higher resolutions and GPU memory usage as the current limitations are shown. 3-D mammography as a method for breast cancer screening covers the second scenario. While specific filters are combined with analytical reconstruction in current systems the realization of advantages of statistical reconstruction methods over the current filtered back-projection methods in this field is demonstrated. The performance measurements underpin the possible application as well as the necessity of GPU acceleration.

Chapter 6 – Summary and Outlook

In the last chapter this thesis is summarized first. Before the possible usage of GPUs in future medical image processing systems is discussed, the Open Computing Language is compared to CUDA for the two major operators. Finally the prevision of future computing trends – such as converging of CPU and GPU technologies – in relation to medical image processing is stated in the Outlook.

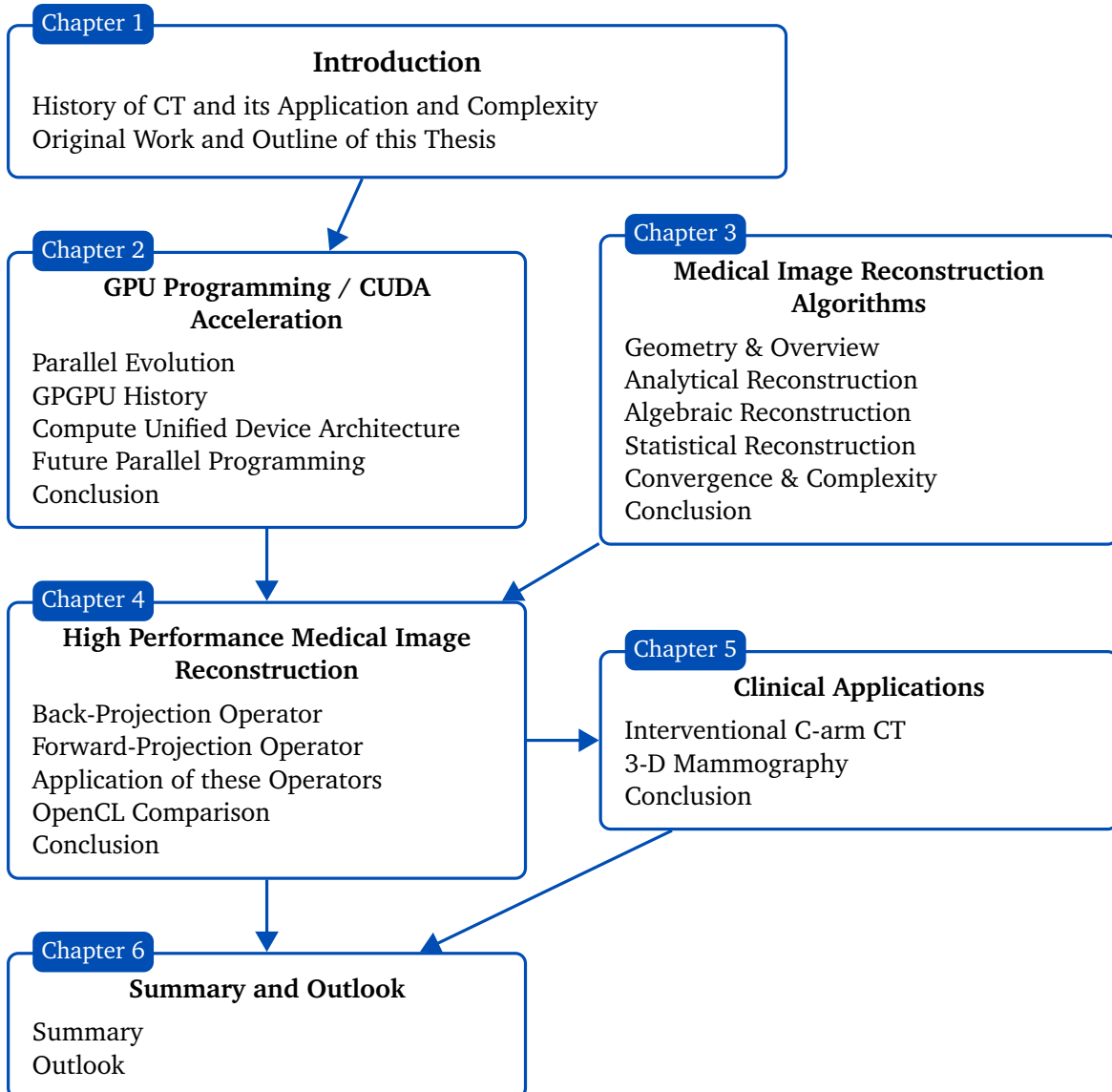


Figure 1.8: The organizational structure of this thesis gives a summary of the different topics of each chapter. The different premises in each chapter are indicated by the directed arrows.

GPU Programming / CUDA Acceleration

Graphics processing units build a strong influence on modern high performance computing. It is likely that GPUs will enlarge their share in compute demanding applications even more over the next years. The basis of our research besides medical image reconstruction algorithms requires knowledge about GPU programming. In the following chapter, the reader will be introduced to the evolution of GPUs used for general purpose computing. Goal of this chapter is to transfer the necessary knowledge for an easier perception of our suggested implementations. For more detailed information we suggest the online lecture “Programming Massively Parallel Processors with CUDA” at Stanford University by Hoberock and Tarjan [Hobe 10] available via iTunesU¹ or refer to the book “Programming Massively Parallel Processors” by Kirk and Hwu [Kirk 10]. The official CUDA programming guide [NVID 07a] and CUDA manual reference [NVID 07c] may be self-evident. After stating a brief history, the important steps in the significant change of GPU Programming are clarified.

2.1 Parallel Evolution

Gordon E. Moore – cofounder of Intel – was quoted as saying that “The number of transistors on an integrated circuit doubles every two years” also known as Moore’s Law [Inte 05]. This law is somehow proven for the last four decades (1970-2010).

After the turn of the millennium serial performance improvements by higher frequencies came to an end. Due to the non-linearly scale of power consumption as well as arising heat, CPU inventors could not continue to increase the frequency in order to not melt the chip. However Moore’s Law still holds as transistors density kept increasing over time. Instead of higher frequencies, manufactures had to improve different levels of parallelism, for instance data-level parallelism. For data-level parallelism a specific instruction for several data elements is not performed serialized but in parallel instead. Most common representatives are vector units for SIMD (Single instruction, multiple data) execution, Streaming SIMD Extensions (SSE) , the CELL Single Processing Element (SPE) and GPUs. Another trend were improvements in thread-level parallelism. Here, different programs respectively threads are not executed alternating on the same executions units, but

¹<http://www.apple.com/education/itunes-u/>

in parallel on different execution units instead. Not only the multi-threading capability was increased, but also the amount of cores respectively compute units. Examples are multi-core and many-core architectures like Intel Core2, Intel Core iX, AMD Phenom, IBM CELL, Sun Niagara or NVIDIA GPUs.

2.1.1 Immense Compute Power

To answer the question, why massively parallel processing is that important these days, somebody can argue very simple: “Because of the immense compute power!” In fact, looking on the historical single- and double- precision peak computing rates – shown in Figure 2.1 – GPUs clearly dominate this field. But the correct answer is more complex than that.

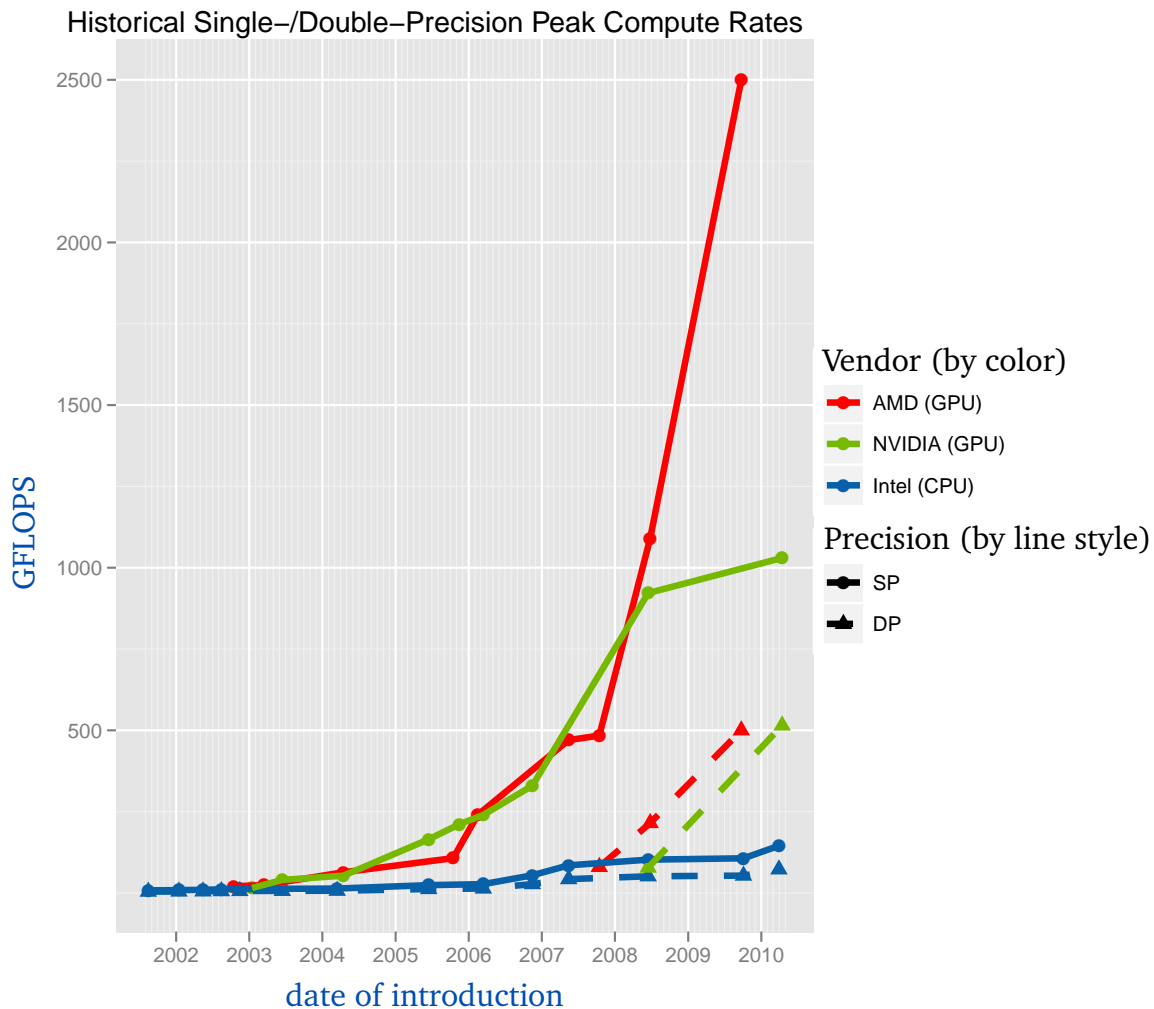


Figure 2.1: Historical single- and double- precision peak computing rate for Intel CPUs and GPUs by AMD and NVIDIA. The plot shows the contrast between CPU and GPU compute performance development over the last decade. The measurements are given in Giga Floating Point Operations per Second (GFLOPS). Courtesy of John Owens.

In 2005, Owens *et al.* already compared [Owen05] the performance for GPUs and CPUs before the introduction of a unified shader architecture. The authors also detailed the limitations and difficulties at that time for General Purpose Computation on Graphics Processing Units (GPGPU). In the following years both GPU vendors could significantly improve the peak performance of GPUs and could clearly outpace the CPU performance development. Before discussing how this high performance can be actually achieved it is as important to look at the history of memory bandwidth development.

2.1.2 Extreme Memory Bandwidth

While GPUs gain their immense compute power due to a tremendous amount of Arithmetic Logic Units (ALU), this logic cannot be reasonably used without the data to be computed. In a realistic scenario high compute performance correlates also to data access performance. In case of massively parallel GPU programs the memory bandwidth is crucial, too.

Analogously to the peak compute rates, the historical peak bandwidth shown in Figure 2.2 proves a significant advantage by the GPU compared to the CPU. On the other hand to conclude, that highly optimized CPU programs are limited by the memory bandwidth is practically wrong. CPUs have a far more advanced technical structure in different levels of cache. In fact it is very hard to get an optimized program on a multi-CPU multi-core system to be limited by the system's memory bandwidth and not the compute barrier.

An example for bandwidth limitation in medical image reconstruction can be found in Hofmann *et al.* [Hofm09a, Hofm10b]. For a system equipped with four Intel Dunnington CPUs (Hexacores) a speedup of 19.5 could be achieved for an optimized single-threaded program compared to 24 threads. This indicates that the system was limited by its bandwidth. But modern CPU architectures like the Intel Dunnington in contrast can scale better than just a linear improvement correlated to the number of cores. A system equipped with two Quad-Core CPUs reached a speedup more than 10 comparing the optimized single-threaded version against 16 threads – making use of Intel's Hyper-Threading Technology (HTT). Looking at the GPU, bandwidth limitation certainly matters. This strongly motivates higher bandwidth rates for GPUs.

2.1.3 CPU versus GPU

It is clear that graphics processing units differ highly from central processing units. As the name central processing states the CPU is the center of data processing. CPUs must be good at everything, even if parallel or not. While the complexity of operating systems grew over time so did CPUs to accomplish the complicated execution of hundreds of different tasks almost at the same time. Therefore the

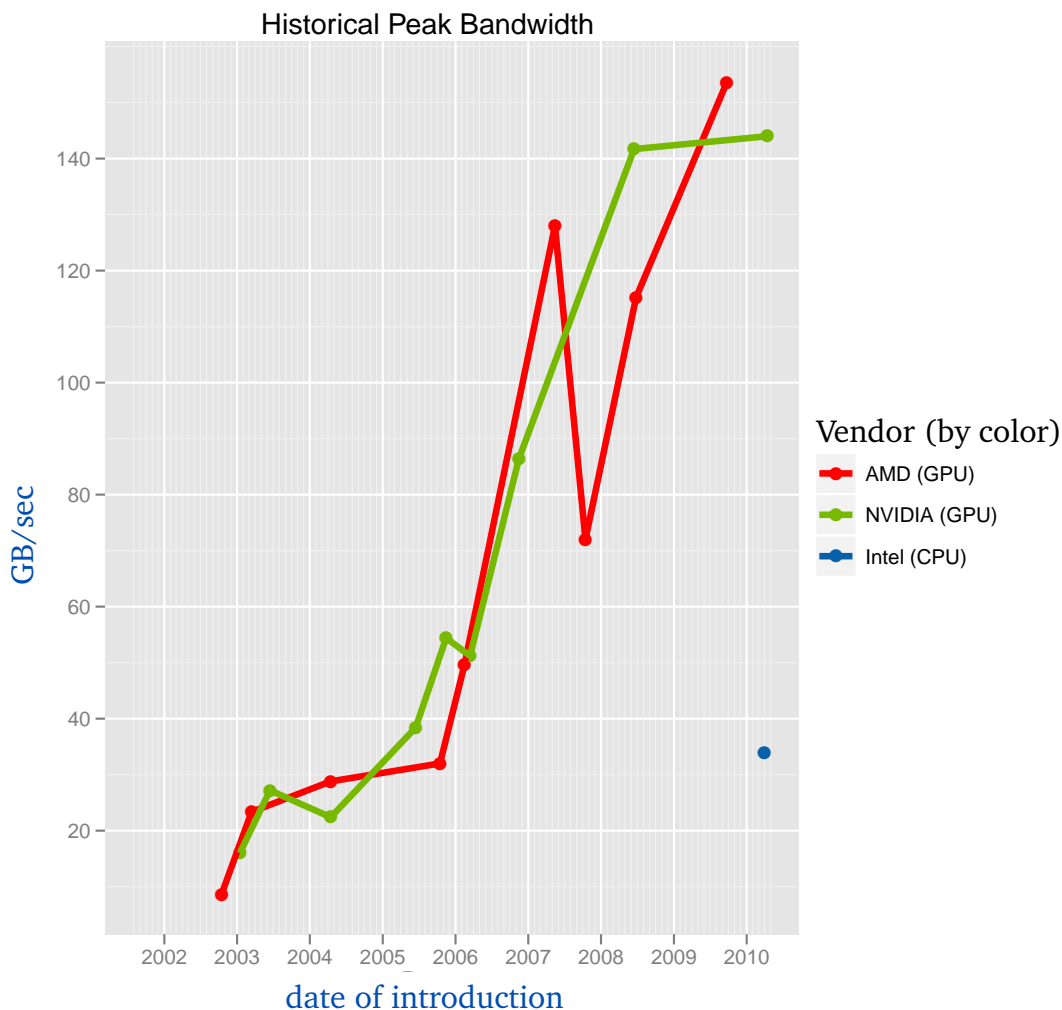


Figure 2.2: Historical GPU peak memory bandwidth for AMD and NVIDIA GPUs. The plot illustrates the immense bandwidth improvements over the last decade. In contrast the maximal memory bandwidth of an Intel CPU is exemplified. Courtesy of John Owens.

latency experienced by a single thread had to be minimized. To do so manufactures introduced big on-chip caches and developed a sophisticated control logic.

Different requirements produce different chips – see Figure 2.3 for a rough visual comparison of the CPU versus GPU. From the traditional graphics pipeline GPU architects learned that throughput is of key importance. The graphics cards respectively the GPU must paint respectively compute every pixel within each frame in a specific time. This specific time defines the frame rate, meaning the amount of frames painted per second. Interactive programs require a frame rate about 5 – 6 frames per second (fps). To achieve real-time a frame rate greater than 20 fps is necessary. While the human brain is limited in its recognition a frame rate of approximately 60 fps is favored for computer games – making the different frames indiscernible for the player.

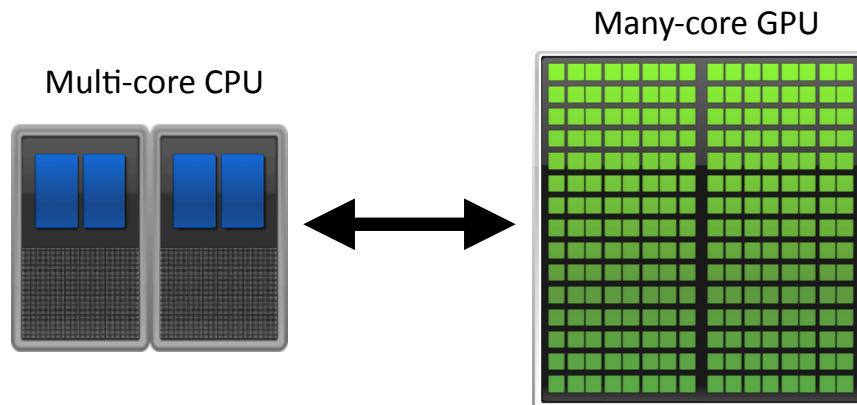


Figure 2.3: CPU versus GPU architecture. Courtesy of NVIDIA.

In order to maximize the throughput GPU developers pursued a different strategy. On the GPU lots of threads are created, run and retire very rapidly. While the number of threads in flight – meaning the number of threads that are actually running and those ready to run – is limited by resources such as registers and bandwidth, lots of resources were built in GPUs in order to gain a high number of threads. The usage of multi-threading allows hiding memory latency. This means that it is accepted that one thread stalls if hundreds are ready to run. Thus, GPU architects skipped big caches and introduced shared control logic across many threads.

Over time motivated by more and more complex graphics of computer games the GPUs gained more and more compute power. More important, specifically programmable hardware components were introduced – starting in 2001 with the NVIDIA GeForce 3 – to realize complex and individual graphics computations. While computer graphics benefited from the increasing compute power, general-purpose computations hardly could.

2.2 GPGPU History

In order to understand General Purpose Computation on Graphics Processing Units, traditional architectures and methods like the graphics pipeline, OpenGL and shading languages are revisited. Afterwards the Compute Unified Device Architecture is explained including the device, execution and memory model.

Until 2006, programmers had a hard time to use GPUs for general purpose. The equivalent of the graphic application programming interface (API) had to be used to access the processors cores – meaning that OpenGL or Direct3D techniques were creatively used to program these chips. This techniques are called GPGPU programming.

Even with a higher level programming environment – e.g., BrookGPU² – the underlying code is still limited by the APIs. The APIs limit the complexity of applications that can be written for these chips. Only a few people overcame the required qualifications to use these chips for a high performance achievement in a limited field of applications. Therefore, GPGPU did not become a widespread programming phenomenon in the early days.

2.2.1 Programmable Graphics Pipeline

Due to the introduction of programmable segments in the graphics pipeline, graphics programmers were able to model a specific behavior in the graphics computation. For example to incorporate shading methods, where the reflection of light from surfaces and the corresponding pixel colors are estimated by interpolating surface normals across rasterized polygons [Shir 02, Enge 06]. In case of the traditional graphics pipeline – depicted in Figure 2.4 – this shading is implemented in the Programmable Fragment Processor. Despite the fragment processor the Vertex Processor provided programmability for specific geometrical transformation and lighting. Both are part of the fixed-function pipeline, which represents the non-unified device architecture.

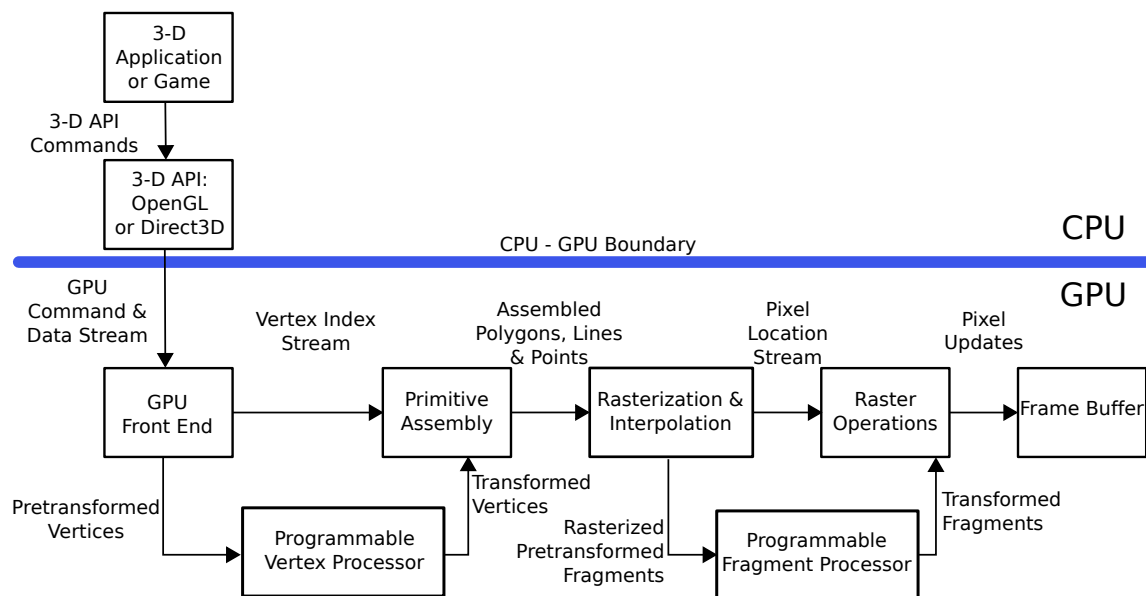


Figure 2.4: Traditional graphics pipeline depicts the fixed-function pipeline despite the programmable vertex and fragment processor. The CPU - GPU boundary illustrates the physical separation of the computation on the graphics card [Fern 03].

Looking at the GPGPU techniques, a programmer had to cast the algorithmic problem into native graphics operations to access the computational resources. Otherwise the computation could not be launched through the OpenGL Shading Language (GLSL) or DirectX using the High Level Shading Language (HLSL). For

²<http://graphics.stanford.edu/projects/brookgpu>

example, to perform a parallelized addition of two matrices, the computation had to be written as a pixel shader. The collection of input data – in this case the two matrices – had to be stored in 2-D texture images and issued to the GPU by submitting a rectangular shape. The result had to be cast as a set of pixels generated from the raster operations and stored in a 2-D frame buffer. The frame buffer is a memory buffer containing the complete data of the frame.

The GPU processor array and frame buffer memory interface were designed to process graphics data. For general numerical applications this is too restrictive. In particular, the output data of the shader programs are single pixels whose memory locations have been predetermined. Thus, the graphics processor array is designed with very restricted memory reading and writing capability. More importantly, shaders did not have the means to perform writes with calculated memory addresses to memory. The only way to write a result to memory was to emit it as a pixel color value, and configure the frame buffer operation stage to write to the frame buffer. The only way to get a result from one pass of computation to the next was to write all parallel results to a pixel frame buffer, then use that frame buffer as a texture map input to the pixel fragment shader of the next stage of computation. Furthermore, no user-defined data types were available. Therefore, most data had to be stored in one-, two-, or four-component vector arrays.

In order to sum up, mapping general computations to a GPU in this early stage of the GPGPU era was quite complicated. Nevertheless, researchers demonstrated a handful of useful applications with the cost of high implementation efforts.

2.2.2 Load Imbalance

At last, another reason for a new device architecture was motivated by the ratio of vertex and fragment shader compute power. Due to the fact that different graphics applications require a different amount of vertex and fragment computation an imbalance existed.

The GeForce 7800 GTX, for example, provided eight vertex shaders and 24 pixel shaders designed on the chip. For specific computer games a high amount of vertex computations is required due to complex scenes with many objects. On the other side specific applications – e.g., virtual reality – seek for realistic effects as good as possible. Therefore high fragment performance is needed. This resulted in an imbalance of the utilized shaders as either vertex or pixel shaders were not fully utilized. The imbalance represents a case depending waste of computing power left on the device. In order to solve this problem a new compute unified device architecture was developed.

2.3 Compute Unified Device Architecture

The field of GPGPU computations changed at the turn of the year 2006/2007. In November 2006 NVIDIA introduced its G80 chip. The first of its kind providing a unified shader architecture shown in Figure 2.5. Instead of separate vertex and fragment shaders this chip provided 16 streaming multiprocessors (SM) each equipped with 8 single execution units – also called scalar processors (SP). The figure illustrates 8 Texture / Processor Clusters (TPC) each consists of 2 streaming multiprocessors (SM) which share 4 texture processing (TP) units also named texture mapping units (TMU). The instruction and data flow as well as different cache levels are additionally illustrated together with the 6 memory controllers.

Overall this chip provides 128 scalar processors running at the shader clock frequency of 1.35 GHz in the first available card – the GeForce 8800 GTX. Thus the introduced NVIDIA GeForce 8800 GTX graphics card achieved a theoretical peak computation performance of 345.6 GFLOPS. This was about 65% more than the – at this time leading – Cell Broadband Engine Architecture (CBEA) providing a peak performance of 208.4 GFLOPS. But for GPGPU the innovation was not only based on the hardware architecture. Together with the architecture a new approach was introduced on the programming side. In order to introduce this approach this is split into three different sections on the programming, the execution and the memory model.

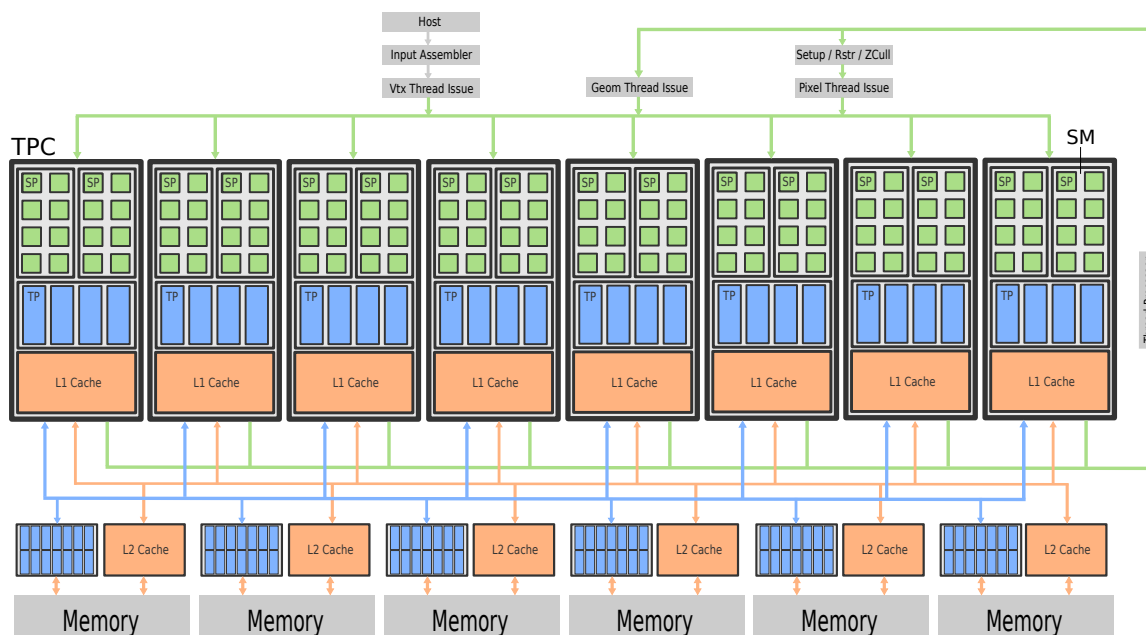


Figure 2.5: The NVIDIA G80 architecture as first graphics processing unit providing a unified device architecture. The sketch is based on the published idea of NVIDIA. The architecture is motivated by the unified shader design.

2.3.1 Programming Model

In 2007 NVIDIA released CUDA – standing for Compute Unified Device Architecture. NVIDIA actually devoted silicon area to facilitate the ease of parallel programming [Kant 08]. Without public notice, NVIDIA had added additional hardware to the chip despite the change in GPGPU software alone. In the G80 and its successor chips for parallel computing, CUDA programs no longer go through the graphics interface at all. Instead, a new general-purpose parallel programming interface on the silicon chip serves the requests of CUDA programs. Moreover, NVIDIA had to redo all other previous software layers to use this chip as well as a high end graphics device. The best thing about it, NVIDIA provided a new API to programmers enabling the use of familiar C/C++ programming tools for parallel programming.

The concept behind CUDA is to program a parallel processors in order to achieve high performance as well as keeping higher functionality and maintainability. More important, the API should be usable not only for NVIDIA's high-end graphics cards, but for NVIDIA's mobile and cheaper low-end GPUs as well as faster future generations. To achieve such a scalable parallel programming model, CUDA introduces a key parallel abstraction.

An easy way to explain this abstraction is in a two-dimensional data-parallel fashion. Assuming that a problem can be computed independently for each element of a two-dimensional array, each element of this array then corresponds to a thread.

Threads

Characteristically for such CUDA threads are lightweight and non-complex computations. A simple example would be adding up 2 two-dimensional arrays. This corresponds in two data elements that are added up for each thread. All threads execute the same sequential program – also called kernel function. In order to differ from each other, each thread can be identified via its thread ID. Due to this ID – a 3-dimensional vector – the data parallelism can be evoked. While each thread executes the same kernel, the data elements are loaded depending on the thread ID. Using this kind of parallelism, a natural way is provided to invoke computation across the elements in a domain such as a vector, matrix, or volume. To deal with more difficult problems as well as to gain scalability these threads are clustered in thread blocks.

Thread Blocks

Threads are grouped into thread blocks. Particular parallel computation problems can be realized in much better performing solutions if the parallel tasks can communicate with each other. Examples for such problems are the histogram computation, the FFT and the computation of an integral, sum or norm of a large vector. Thread blocks have a block ID. The combination of block ID and thread ID provides a unique identification for each of the parallel computations. An automatic mapping of parallel programs – providing thousands of threads – onto hardware is a fairly hard task. CUDA virtualizes the physical hardware. For the programmer the scheduling of these thread blocks is unknown on purpose. Thread blocks should be independent, since they will be scheduled onto physical hardware without specific

priorities. The execution of thread blocks and their threads is explained in the next subsection where the execution model is covered.

Grid

In order to complete the key parallel abstraction, all thread blocks are stacked in a grid. This implies that the whole parallel computation consists of one grid filled with thread blocks – made of lightweight threads. Due to this abstraction CUDA achieves its scalability for different and future GPUs by NVIDIA. An overview of this concept is depicted in Figure 2.6.

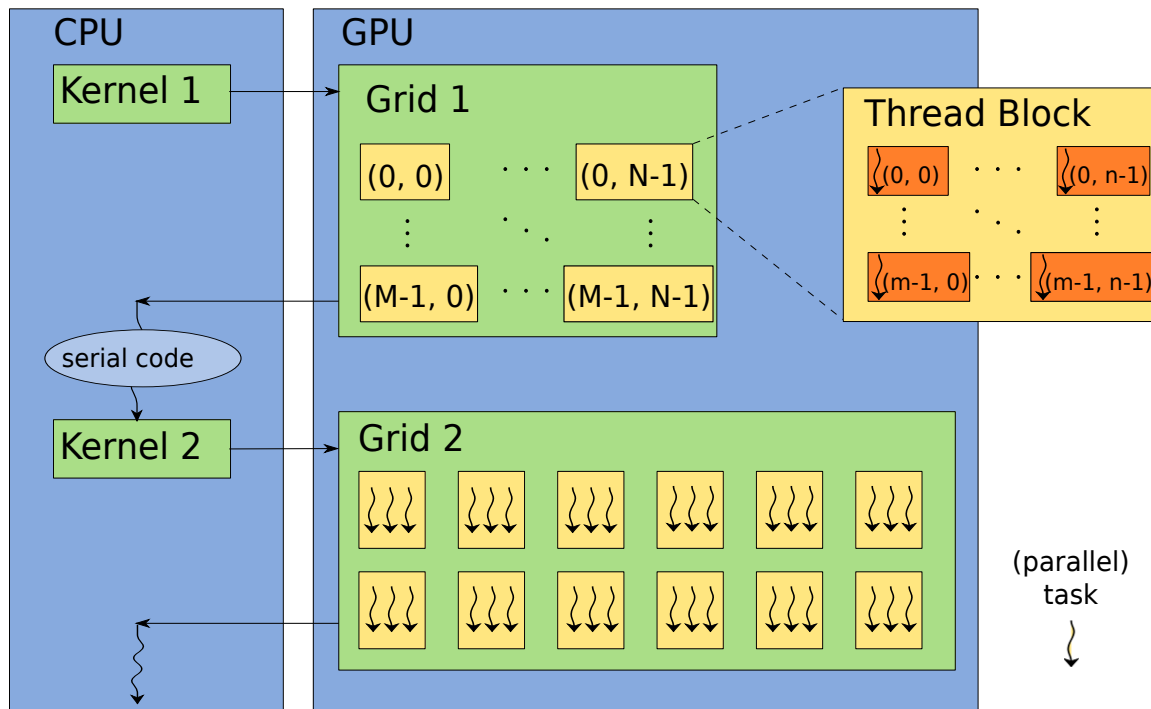


Figure 2.6: Parallelization by a grid and block scheme keeps scalability and hardware independence. The execution is split between serial CPU code as well as parallel GPU code. The snaked arrows each represent a (parallel) task. The sketch is based on the idea of NVIDIA [NVID 07a].

Invoking this parallel scheme, the CUDA API encompasses a whole software environment. CUDA C is a variant of C with extensions. It is compiled with NVIDIA's compiler – *nvcc*. The extensions can be grouped into four different categories [Kant 08]:

- Function qualifiers: indicating whether a function executes on the host CPU or the GPU
- Memory type: indicating where a variable is located in the GPU address spaces
- Execution configuration: specifying the execution parallelism of a kernel function in terms of grids and blocks
- Unique identifier: state variables which store grid and block dimensions and thread IDs

The CUDA programming language is made for a heterogeneous serial-parallel computing. Parts of the program are executed serially on the CPU while the decomposed parallel threads are accelerated by the GPU. CUDA provides a straightforward mapping onto the hardware that fits well on the GPU architecture. But this model maps as well to multi-core CPUs, which will be discussed together with future parallel programming in Section 2.4.

In order to achieve a hierarchy of concurrent threads, it is important that a programmer identifies the inherent parallelism of the problem to be solved respectively the algorithm to be executed. The challenging part is to figure out a lightweight decomposition that is parallel at a large scale. The program then benefits from the GPU's immense computer power if the program meets the execution model.

2.3.2 Execution Model

One of the key architectural ideas is inside the Streaming Multiprocessor (SM). NVIDIA describes its SM to be likewise a SIMD unit respectively vector unit. Their architecture is called SIMT, standing for Single Instruction Multiple Thread. Analogously to SIMD each scalar processor computes the same instruction at a specific time. Differing from SIMD, NVIDIA wants to indicate by SIMT that the vector size is hidden from the programmer. The vectors are automatically filled and handled by the hardware.

The frequency domain of the streaming multiprocessor is split into the chip- and shader-frequency. The SPs are running at the shader frequency – also called “hot clock” – almost twice as fast as the chip clock. The instruction decoding, registers and data handling are processed with the slower chip clock. Instructions like floating-point multiplication, addition as well as fused multiply-add and integer addition demand four clock cycles to be computed. Floating-point division, modulo, integer multiplication and division require more cycles. More complex instructions are replaced by combining several basic instructions.

While the G80 and its successor chips are based on a pipelining principle, two instructions can be issued in every chip clock cycle. In the ideal case an instruction is computed in four shader clock cycles respectively two of the chip clock. Assuming a filled pipeline, several four cycle instructions can be streamed by instruction planning. This results effectively in a computation time of only one shader clock cycle for those instructions. This principle and the interaction between chip- and shader clock is illustrated in Figure 2.7. The peak performance then can be computed for the case of using only fused multiply-add instructions – e.g., matrix multiplication.

The CUDA execution model maps the parallel abstraction to the silicon. Threads within a thread block run in groups of 32 called warps. These threads in a warp share the same instruction units. In order to hide latencies, the hardware relies on threads. While a specific warp is waiting for data, any other warp – not waiting for anything – can run instantly. The hardware is designed that a context switch is basically free. Additionally lightweight synchronization primitives are provided to keep the abstraction model upright.

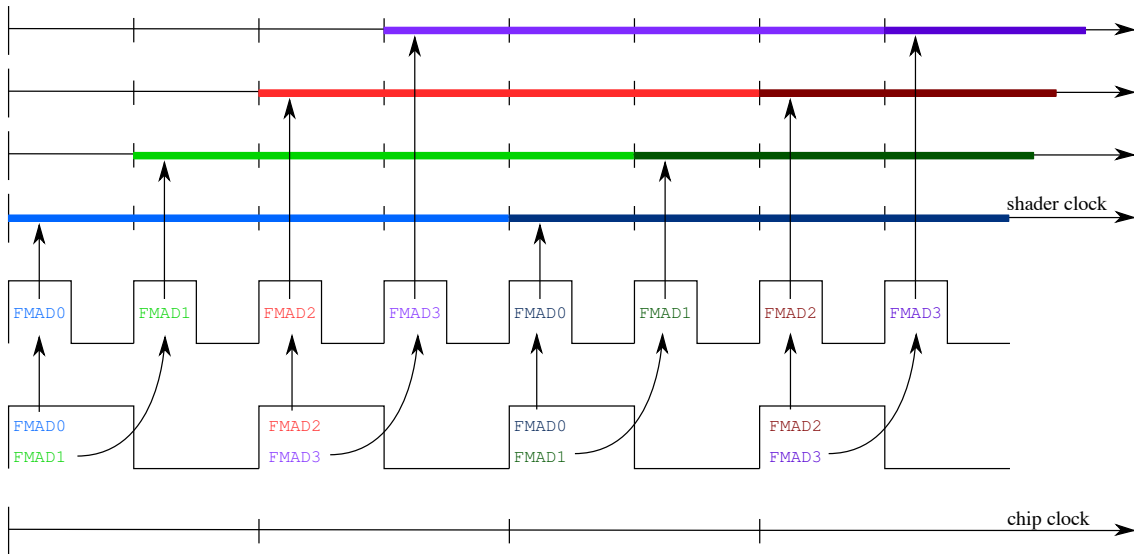


Figure 2.7: Ideal instruction planning. The different colored bars depict different instructions and their execution on a single Arithmetic Logic Unit (ALU). The fused multiply-add (FMAD) operations are executed respectively combined together in the ideal pipeline principle. Therefore at each cycle of the shader clock a FMAD operation finishes while four of them are executed in parallel. The sketch is based on the description of NVIDIA [NVID 07a].

It is assumed, that there is always certain work available to execute. Since there are no branch predictors, warps which execute a branch wait until every thread in the warp has been calculated. When a warp diverges – threads within a warp are executing different instructions – performance gracefully decreases. If there are N divergent paths in a warp, performance decreases by about a factor of N , depending on the length of each path. Each divergent path of a warp is serially issued until all threads can continue with the same instructions. In contrast, it is not an issue if all threads within different warps execute a different branch. An example for branching and serialization is illustrated in Figure 2.8.

In summary, NVIDIA describes their execution model as Single Instruction, Multiple Thread (SIMT) a variant on SIMD. From a programming perspective this is big difference. The vector width is architecturally visible for SIMD and data must be packed and unpacked into vectors for computation. In the SIMT model, execution width is a micro-architectural feature handled solely by hardware and a SIMT instruction such as a conditional branch specifies the behavior of a single independent thread. A thread is a virtualized scalar processor (registers, program counter, state). A thread block is a virtualized multiprocessor (threads, shared memory/-cache). Threads and thread blocks launch and run to completion. Having such a flexible and powerful execution model is important as well as accessing the data to be computed. The latter is covered in the memory model.

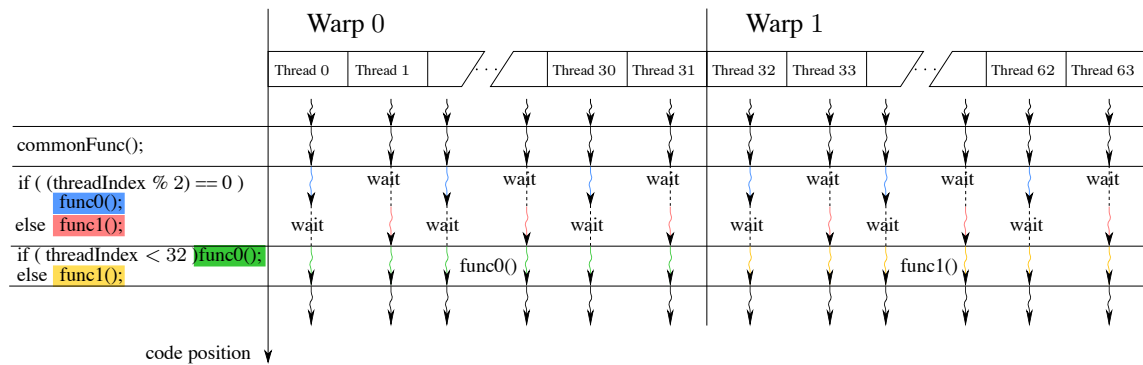


Figure 2.8: This example illustrates the execution of threads within a warp in case of divergence. While the *commonFunc* can be executed in parallel, half of the threads in both warps have to wait for each conditional branch and the performance decreases by a factor of two. This is illustrated by *func00* and *func10*. Instead if the condition is warp dependent – equal for all 32 threads within a warp – again the parallelization can be fully utilized. This is illustrated by *func00* and *func10*. The sketch is based on the description of NVIDIA [NVID 07a].

2.3.3 Memory Model

In addition to an execution model, CUDA also specifies a memory model – shown below in Figure 2.9. It provides a variety of different address spaces for communication within the GPU and with the host CPU. Fast, on-chip storage is colored in red. The memory kept in the DRAM of the graphics device is depicted in grey. The important attributes are stated – after detailing each memory – in Table 2.1.

Registers

The largest and the fastest level of the on-chip memory hierarchy is the register file. It provides 32 – 64 kB space on the chip. Register-to-register instructions achieve the peak instruction throughput if the vector length is large enough. Equally to a CPU register file, it is private for each thread and read-/write-able. Typically the variables, input and output operands for a thread are stored in registers. The amount of registers is limited depending on the occupancy, the kernel complexity and the GPU generation. Should the register file be exhausted, then data spills into local memory.

Local Memory

Like registers, local memory is private for each thread, but is held in DRAM. Therefore its performance is equally slow as global memory and the usage should be avoided. It is introduced to provide a dynamic approach of register files in order to overcome hard limitations. Otherwise the complexity of kernel functions would be dramatically limited in general. The price to be paid is performance loss, in case it's utilized. Instead programmers should use shared memory.

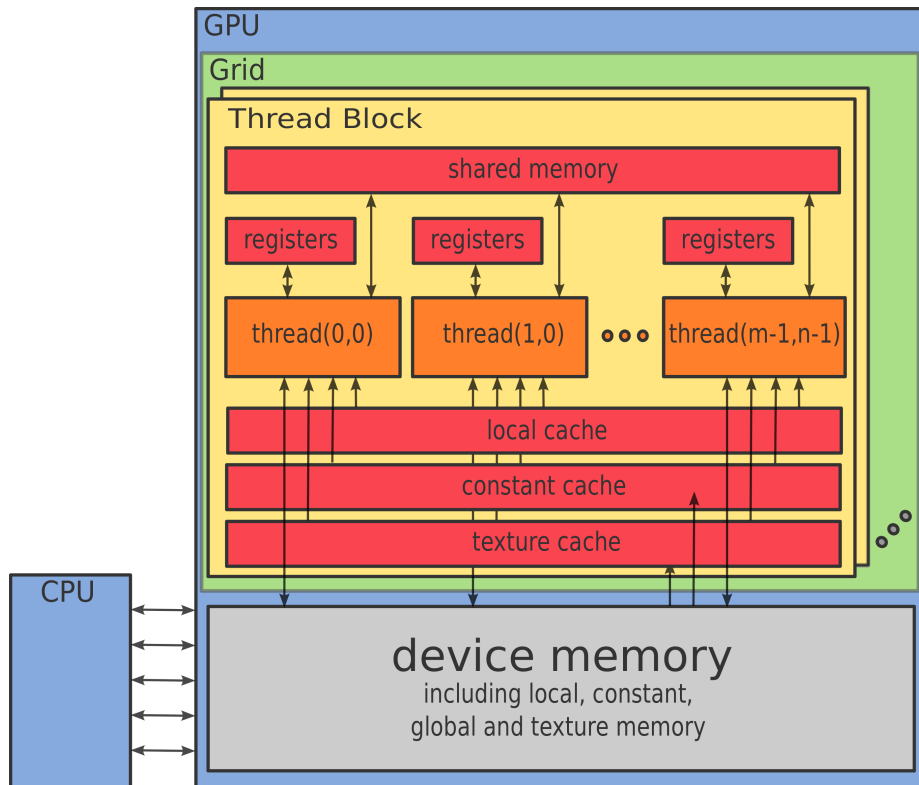


Figure 2.9: The CUDA memory model. The sketch is based on the description of NVIDIA [NVID 07a].

Shared Memory

The shared memory can be used for communication between all threads of a thread block as well as primary local storage space. Even it is on-chip memory, it is slightly slower than register files [Volk 08a]. Shared memory is generally the lowest latency communication method between threads. Shared memory can be used for a variety of purposes, such as holding shared counters or shared results from thread blocks. It is read- and write-able, but no coherency is guaranteed if two threads try to access it at same point of time. Therefore atomic functions are included in the framework. On the first and second generation GPU only 16 kB of shared memory are provided per SM and shared for all thread blocks running on the SM. In order to achieve low latency for all threads accessing this memory, bank conflicts must be avoided.

Constant Memory

The constant memory is one of the read only address spaces. With 64 kB it is a relatively small space and can be used, e.g., for random accesses and frequently-accessed parameters inside the kernel function. The memory resides in the DRAM, but since it is read only, it is readily – if cached – on-chip. The constant cache does not enforce coherency. Thus if the CPU writes to the constant memory, the caches are invalidated before using the new data. This is equally for texture caches.

1-D Texture Array

A one dimensional texture array in CUDA is basically similar to constant memory. The big difference is the allocation and initialization as well as the usage inside the kernel function. To access a 1-D texture array a specific function – *tex1D(textureRef, float pos)* – has to be used. In contrast to the constant memory the texture array allows an automatic interpolation between neighboring values – in hardware – depending on the given position. All textures in common is, that the texture function implies an addressing mode as well as specific data formats – e.g., for the RGBA color representation of a pixel. The addressing mode specifies how out-of-range texture coordinates are handled – e.g., clamped to the texture border. The size of 1-D texture arrays is limited to 8192 elements residing in DRAM.

1-D Linear Texture

The difference between a 1-D linear texture and a 1-D texture array is the missing texture interpolation. Therefore to access a 1-D linear texture the function call uses integer positions – *tex1Dfetch(textureRef, int pos)*. In contrast to the 1-D texture array, the 1-D linear texture is write-able for kernel functions. Since the texture caches don't enforce coherency, it is of importance to understand that the behavior is undefined if a thread writes to a certain position during an kernel execution while another thread is reading this position. After an kernel execution the caches are invalidated such that a 1-D linear texture can be either used as write-only or read-only during a kernel execution. 1-D linear texture can contain up to 2^{27} elements respectively 512 MB for float values.

2-D Texture Array

Analogously to the 1-D texture array, the two-dimensional provides read-only data, but provides a bilinear interpolation by hardware. In order to access the 2-D array a different function – *tex2D(textureRef, float posX, float posY)* – is used. Clamping and cache are supported as well. A little bit tricky is the size limitations, since the maximal 2-D texture size is defined by 2^{16} bytes $\times 2^{15}$ elements. This means that depending on the data type 64 kB can be used in the first dimension while the second dimensions is limited by 2^{15} elements. Overall a maximum of 2 GB per 2-D texture array can reside in DRAM.

2-D Texture from Pitch-Linear Memory

At last with CUDA 2.3, NVIDIA introduced 2-D textures from pitch-linear memory. In principle they are not distinguishable from 2-D texture arrays, despite the fact they are write-able by the kernel. Like 1-D linear textures, the cache is not providing coherency, such that the behavior for reading and writing during a kernel execution is undefined. However, they can be used either for writing or reading during a kernel execution. Access by function *tex2D*, size limitations as well as bilinear interpolation by hardware are equal to 2-D texture arrays.

3-D Texture Array

The capability of 3-D texture arrays was introduced in CUDA 2.0. It is the three dimensional companion piece to two dimensional texture arrays: Not write-able via kernels, cached, and providing trilinear interpolation. However the actual data layout on the device is swizzled. Swizzling [Enge06] is originated by computer graphics in order to improve the average sampling rate for different view directions. This ensures a balanced memory access performance. In Chapter 4 the size limitations for 3-D texture arrays is again in focus due to its restrictions. A maximum of 2048 elements in each direction is the boundary for such 3-D texture arrays in CUDA. Therefore theoretically a volume made of 2048^3 elements – 32GB float volume – can be stored in DRAM. However, current graphics cards do not provide this enormous amount of memory. Unfortunately, there is no 3-D write-able texture available up to version 2.3 of the CUDA API.

Global Memory

The global memory completes the CUDA memory model. As the name states it is globally visible to an entire grid and can be arbitrarily written to and read from by the GPU or the CPU. Since the global memory can be written to, it is not cached anywhere on chip. Compared to registers, shared memory and textures it is the slowest possible access. The non-cached memory access takes about 470 – 720 cycles [Volk08a], roughly matching the official 400 – 600 cycle stated by NVIDIA. Global memory resides in DRAM and is limited only by the amount of memory available on the graphics card. Volkov *et al.* made an detailed analysis [Volk08a] of CUDA's memory model.

memory type	speed	write-able	caching	hw. interp.	size limit
registers	+++	yes	no	no	by GPU
shared memory	++	yes	no	no	by GPU
constant memory	+	no	yes	no	64 kB
1-D texture array	+ / -	no	yes	yes	8192
1-D linear text.	+ / -	either	yes	no	2^{27}
2-D tex. array	+ / -	no	yes	yes	64 kB × 32 k
2-D tex. pitchl. m.	+ / -	either	yes	yes	64 kB × 32 k
3-D tex. array	+ / -	no	yes	yes	2048^3
local memory	-	yes	no	no	by device
global memory	-	yes	no	no	by device

Table 2.1: Overview of the memory types available in CUDA. This includes a speed – cache depending – review, the capabilities of writing via threads as well as caching and hardware interpolation.

The reviewed memory model shows advantages as well as disadvantages of the utilized GPU memory. Limitations and missing features – respectively later intro-

duced memory support – are of importance. Without detailed knowledge about this memory model a parallel implementation is still possible, but a huge loss in performance is very likely.

2.3.4 Utilized CUDA GPUs

The hardware side of the equation is equally important. For our research, development as well as evaluation and performance measurements, we utilized different kind of graphics cards since 2007. In the following we shortly want to give an overview and detail important properties. Table 2.2 shows the utilized graphics cards including GPU generations and frequencies for chip-, shader- and memory-clock. Later, we will show the significance of available DRAM and memory bandwidth, such this capabilities are stated as well. For the texture performance, we used the numbers officially stated by NVIDIA, except the Performance of the Tesla C1060, which is taken from Schwarz *et al.* [Schw 11] ³

graphics cards	GeForce 8800 GTX	QuadroFX 5600	GeForce 8800 Ultra	GeForce GTX 280	Tesla C1060
GPU gen.	G80	G80	G80	GT200	GT200
shared mem.	16 kB	16 kB	16 kB	16 kB	16 kB
chip clock	575 MHz	600 MHz	612 MHz	602 MHz	612 MHz
shader clock	1350 MHz	1400 MHz	1500 MHz	1296 MHz	1296 MHz
db. memory cl.	900 MHz	800 MHz	1080 MHz	1107 MHz	800 MHz
# of SMs	16	16	16	30	30
# of SPs	128	128	128	240	240
register file	32 kB	32 kB	32 kB	64 kB	64 kB
texture units	6	6	6	8	8
memory BW	86.4 GB/s	76.8 GB/s	103.7 GB/s	141.7 GB/s	102.4 GB/s
GTexels	36.8	38.4	39.2	48.2	48.8 ³
DRAM	768 MB	768 MB	1.5 GB	1 GB	4 GB

Table 2.2: Overview of technicals details for the utilized graphics cards.

For the research in this thesis the 1st and 2nd generation of CUDA capable graphics cards were utilized. The 3rd generation – namely “Fermi” architecture – is shortly introduced in Section 2.4.4, but not evaluated. Neither the presented code is optimized for that architecture. As GPU architectures currently evolve about every two years, newer GPU architectures are out of scope in this thesis. The first generation is based on NVIDIA’s G80 chip, the first unified device architecture, where we started our research. Due to the fact that NVIDIA’s GT200 – the 2nd generation – is an incredibly aggressive derivative of the G80 architecture, we extended

³The performance number is taken from [Schw 11]

our research. The GT200 provided substantial improvements in almost every aspect of the architecture [Kant08]. Increasing the overall performance as well as adding more features for programmability furthering NVIDIA's vision of GPGPU. Many changes are obvious, such as the registers per SM, the number of SMs, or the ratio of SMs to memory pipelines. An extensive description about NVIDIA's G80 and GT200 GPU as well as further details about improvements can be found in Kanter *et al.* [Kant08]. With the introduction of the GT200 NVIDIA certainly cemented their status as leader in general purpose computation on GPUs.

2.3.5 Optimizations

In order to gain high performance certain optimizations are considered in our implementations. The performance of GPU implementations heavily depends on an optimal usage of the underlying hardware. In many tutorials, lectures and books a similar statement as follows can be found [Kirk10, p. 15, l. 11]:

“Someone once said that if you don't care about performance, parallel programming is very easy.”

The introduction of CUDA strongly simplified parallel programming as well as GPGPU. Using a few lines of code, simple compute intensive programs can be parallelized and ported to the GPU within few hours. Since the GPU provides substantial compute power, even non-optimized parallel programs can gain a significant performance improvement. Unfortunately, many of these performance improvements are published, giving the delusive impression that using GPU programming an performance improvement, more than hundreds time faster, easily is achieved. Most of the times, the compared CPU programs are totally non-optimized and possibly poorly designed for CPU computation. To get a better impression, we recommend Lee *et al.*, who published [Lee10] a relevant CPU vs. GPU performance comparison for several applications.

However, high performance computing using GPUs is still important and has significant impact especially for medical image reconstruction as shown in this thesis. By augmenting C/C++ with minimalist abstractions, programmers can focus on parallel algorithms and not the mechanics of a parallel programming language. To gain steady performance the program should scale to hundreds of cores and ten-thousands of parallel threads. GPU threads are lightweight, create and switch is free. Therefore the GPU typically needs thousands of threads for full utilization. This is not only important to achieve a high compute rate, but more to hide memory latencies.

As mentioned in Section 2.3.2, a warp diverges if threads within execute different instructions. This branching has to be serialized and performance is decreased. In order to achieve high performance, the programmer has to prevent branching by strategy if possible.

Since computations are performed on data, we distinguish between three types of memory access. Common shared input data – typically a small amount of data,

non-structured input data and output respectively updated data. The different memory types are explained in Section 2.3.3.

In order to read data that is equally used throughout thread blocks the performance is improved by using shared memory for smaller amounts of common data and parameters. Alternatively – limited by registers – kernel parameters can be used if the data equals for all thread blocks.

For data that is non-structured and read only the programmer should utilize textures and their hardware interpolation if applicable. In this case considering the texture caches and cache-lines by specific access patterns additionally improves performance.

For the last type of memory access, output or updated data, global memory has to be utilized. Due to the fact that the global memory bandwidth is the limiting factor in many cases. Coalescing techniques can dramatically influence performance if global memory is used heavily. Basically the favorable access pattern is achieved when the same instructions for all threads in a warp access consecutive global memory locations. Here, the hardware combines, or coalesces, all of these accesses into a consolidated access to consecutive DRAM locations – achieving a data rate close to the peak of global memory bandwidth.

The introduction of shared memory indented to allow communication between threads. However, different program designs may minimize communication between threads and thus improve performance.

Theoretically, an optimal block- and grid-size can be estimated by the problem design, but it turned out that this is not always the case. Therefore, heuristically determining the best performing block- and grid-size for each different device and utilized kernel is recommended.

For further optimization techniques and performance consideration we refer to NVIDIA's "CUDA C Best Programming Guide" [NVID 10a] or the book by Kirk *et al.* [Kirk 10].

2.4 Future Parallel Programming

In the last Section – Future Parallel Programming – we want to focus on several changes in the field of GPGPU and parallel computing, introduced since 2008.

First to be named are two new frameworks for parallel computing. Most promising is the Open Compute Language (OpenCL) administrated by the Khronos Group. After that, Microsoft's Direct Compute is mentioned briefly. As an extension for CUDA, NVIDIA's announced a general x86 compiler for CUDA programs. The compiler is supposed to allow an effective use of CPUs compute performance – including vector units – for parallel CUDA programs. At last, NVIDIA also introduced the third generation of CUDA capable devices. The architecture – named Fermi – is considered and relevant changes are skimmed.

2.4.1 Open Computing Language

In 2007, NVIDIA launched CUDA which is today the most mature of the emerging programming models and toolchains for GPUs. Certainly there are quite a few other efforts in the same field, some pre-dating CUDA and some created in response to CUDA.

An alternative for parallel programming is the Open Computing Language (OpenCL). In order to avoid being tied to any specific hardware vendor, Apple began working on what would become OpenCL. After the initial development, OpenCL was refined in collaboration with technical teams at AMD, IBM, Intel and NVIDIA. The initial proposal was submitted to the Khronos Group, common for the OpenGL standard. Since June 2008 OpenCL is under the auspices of the Khronos Compute Working Group. In the end of 2008 OpenCL finally found approval and was publicly released as an open standard for general purpose parallel programming of heterogeneous systems. Despite CUDA, which is supported merely by NVIDIA GPUs, OpenCL can be utilized also on AMD GPUs, x86 CPUs and CELL processors. Theoretically, OpenCL provides a portability between different multicore/multiprocessor platforms.

CUDA and OpenCL have a lot in common. The first versions of CUDA (up to V3.0) and early versions of OpenCL (up to V1.1) were very similar in terms of functionality, but provided a slightly different nomenclature. The key-concept of OpenCL is a hardware abstraction layer, consisting of a *platform*, *execution* and *memory model*. The platform model is analog to CUDA's device model, which is not discussed. The *execution* and *memory model* are almost the same within CUDA and OpenCL. A list of most important technical terms in comparison between both can be found in Table 2.3. Technical details about OpenCL can be found on the official website⁴ of the Khronos Group, their *Quick Reference Card* [Khro 10] and for the NVIDIA architecture in [NVID 10c].

The three models can be summarized shortly subsequently:

⁴<http://www.khronos.org/opencv/>

Idea	CUDA term	OpenCL term
executing host	host CPU	host
core	streaming multiprocessor	compute unit
processing units	scalar processor	processing elements
program	host thread	host program
parallel program	thread	work-item
cluster of parallel programs	thread block	work-group
organization of these	grid	NDRange
communication memory	shared memory	local memory
	constant memory	constant memory
	texture memory	image memory
local indices	thread index	local id
global indices	block & thread idx.	global id

Table 2.3: Nomenclature similarities between CUDA and OpenCL. The table shows clearly a conceptual analogousness, but the technical realization in software indicates differences in performance.

1. The *platform model* consists of a host connected to one or more OpenCL devices. An OpenCL device is divided into one or more compute units (CUs). These are further divided into one or more processing elements (PEs) analog to CUDA's SP.
2. Like in CUDA, the execution of an OpenCL program occurs in two parts: a host program that executes on the host and kernels that execute on one or more OpenCL devices. The host program defines the context for the kernels and manages their execution. The OpenCL *execution model* is defined by how the kernels execute. When a kernel is submitted for execution by the host, an index space is defined and an instance of the kernel executes for each point in this index space. This kernel instance is called a work-item and is identified by its point in the index space (analog to CUDA thread). Each work-item executes the same code but the specific execution pathway through the code and the data operated upon can vary per work-item. Work-items are organized into work-groups (analog to CUDA thread blocks). The work-groups provide a more coarse-grained decomposition of the index space.
3. Using the OpenCL *memory model* the kernel executing work-items have access to distinct memory regions each with different restrictions and access speed like in CUDA (see Section 2.3.3). However, not all memory types were initially fully supported. Another difference to CUDA is that OpenCL describes platform specific features. For example, the usage of texture arrays is called image object in OpenCL. This feature is not necessarily supported by all devices that supports OpenCL. The developer has to keep that in mind if such features – may giving significant performance benefits – are utilized.

While CUDA is not an open industry standard and does not work with ATI or Intel GPUs, it was readily available and far more mature and programmer friendly than the other alternatives. For that reason, we focussed on CUDA in our research for acceleration of medical image reconstruction. In order to cover the performance difference between CUDA and OpenCL, we refer to the paper [Du 12] of by Du *et al.* and provide an additional performance comparison in the Section 4.3.

2.4.2 DirectCompute

Another important parallel programming API for heterogeneous computing systems is Microsoft's DirectCompute. The API takes advantage of the massively parallel processing power of modern GPUs to accelerate applications in Microsoft's latest operating systems [Kirk 10]. DirectCompute is part of Microsoft's DirectX and was released with DirectX 11 API. The DirectCompute architecture shares a range of computational interfaces with CUDA and OpenCL.

2.4.3 CUDA x86

In September 2010, NVIDIA introduced *CUDA x86* at the GPU Technology Conference (GTC) in San Jose, California. *CUDA x86* was developed in collaboration with PGI (The Portland Group). It enables CUDA applications to be accelerated on basically any PC. In order to provide an alternative for non CUDA-capable systems, the *CUDA x86* compiler utilizes multiple cores and their SIMD units for parallel execution.

Instead of returning that no CUDA device is found, the program decides during runtime to execute on the CPU instead of the (missing) CUDA-capable GPU. Therefore a single application can be used universally on a computer, but provides a possibly higher performance for proper GPUs being present.

The *CUDA x86* is in development, but with it NVIDIA could provide developers a single parallel programming model to target many core GPUs as well as multi-core CPUs.

2.4.4 Fermi

With the consecutive development of the GPU generations, G80 and GT200, NVIDIA clearly stepped forward into the age of parallel computing. Both CUDA generations dominated the GPGPU market and moreover made parallel computing and HPC accessible for almost every developer – since CUDA is limited to NVIDIA GPUs.

From this success and expertise for future demands, NVIDIA developed their 3rd unified GPU architecture, code named “Fermi”. The implementation and optimization for this architecture is beyond the scope of this work and hence we refer to NVIDIA's white paper [NVID 10b] on the “Fermi” architecture for further details. Due to the fact that the architecture reveals a good impression about the development of future GPU generations, we shortly discuss the architecture and relevant changes.



Figure 2.10: NVIDIA's most advanced GPU architecture, "Fermi" [NVID 10b]. Courtesy of NVIDIA.

The first Fermi based GPU architecture was designed to feature up to 512 SPs. The whole chip therefore required up to 3.0 billion transistors. Each of the SPs obtained several enhancements such that a floating point or integer instruction can be executed per clock, pipelined for a thread. For Fermi NVIDIA changed the vector width of its' SIMT model from 8 to 32 in order to provide more performance. The overall 512 SP's are then partitioned into 16 SMs. Also the overall amount of DRAM was increased from 4 up to 6 GB of GDDR5 memory. The chip design is depicted in Figure 2.10. It illustrates six memory interfaces and the host interface. The GigaThread global scheduler indicates the distribution of the thread blocks to the SM thread schedulers.

Massively parallel executed programs often are limited by the memory bandwidth. In this case, the Fermi architecture is of special interest, because of another improvement. Fermi GPUs provide L1 and L2 cache. The 64 kB of on-chip memory for each SM is configurable either to 16 kB shared memory and 48kB L1 cache or the other way around. This provides more flexibility for programmers and improves shared memory intensive applications as well as optimizes global memory accesses. All SMs share the second level of 768 kB L2 cache. This cache level serves all clients – meaning load, store as well as texture tasks.

For simulation and other numerical sensitive applications – depending on double precision – NVIDIA provided a tremendous enhancement. Like for CPUs, double precision computations can be performed up to half the floating point performance. This reflects a performance increase of factor 8 compared to the GT200 chips. How-

ever, in the beginning NVIDIA had problems with the temperature as well as the power consumption of the chip. Therefore the first Fermi chips did not provide the full capacity [Kant 09].

2.5 Conclusion

Modern high performance computing is heavily influenced by GPU technology as it provides extreme computational power combined with extreme memory bandwidth at the time being multiple times faster than CPUs. CUDA dramatically changed GPGPU programming and empowered a broad range of developers to utilize the GPUs' computational power without the previously required knowledge about graphics engines. Of particular importance for this parallel processing technology are the key abstraction concepts of CUDA. The programming, execution and memory models enable a scaleable approach for a variety of NVIDIA GPUs, including future generations which will provide even higher performance. This generic approach allows different implementations for the parallelization of an algorithm, but not necessarily all provide high performance. In order to achieve high performance it is necessary to understand the underlying hardware as well as the exposed technical features of the architecture in detail. Together with the important aspects for performance optimization and future parallel programming, the basics for the CUDA acceleration and GPU Programming are constituted. Thus, the high computational power of NVIDIA GPUs can be utilized using CUDA.

Medical Image Reconstruction Algorithms

Medical imaging technology is not only founded on mechanical devices and electronic systems like detectors and X-ray emitters. For decades, data processing and specifically reconstruction algorithms have been in the focus of still ongoing research and development.

In this chapter we would like to give an overview of the field of reconstruction algorithms and a possible classification into different groups. Afterwards the considered algorithms are introduced in order to provide the theoretical basis of our scientific contribution. We will further utilize at least one algorithm of each introduced group in Chapter 4. At the end of this chapter, we compare and evaluate their complexity to understand the computational differences.

3.1 Reconstruction in General

The problem statement of reconstruction in general is to compute the composition of an object from measurements. In medical imaging these objects are described as a 3-D object, respectively a 2-D slice out of a 3-D object. The measurements for CT are typically generated by a pair of an X-ray emitter and a detector measuring the remaining radiation energy of the attenuated X-rays. For molecular imaging, the measurements are generated by emitting radio-active substances. Different detectors are used due the fact of a differing kind of radiation that is emitted. To complete the main reconstruction fields in medical imaging, the wide field of reconstruction for Magnetic Resonance Imaging (MRI) needs to be named. This type of reconstruction is not covered and hence we refer to the literature.

After describing the general geometry model used in this thesis, we give an overview of the wide field of reconstruction methods with focus on Computed Tomography and group these depending on their background.

3.1.1 Geometry and Acquisition

The geometry model used in this thesis is derived from a 3-D projection, shown in Figure 3.1. The emitted ray intersects the 3-D object consisting of different materials. Each material is characterized by its physical properties represented by

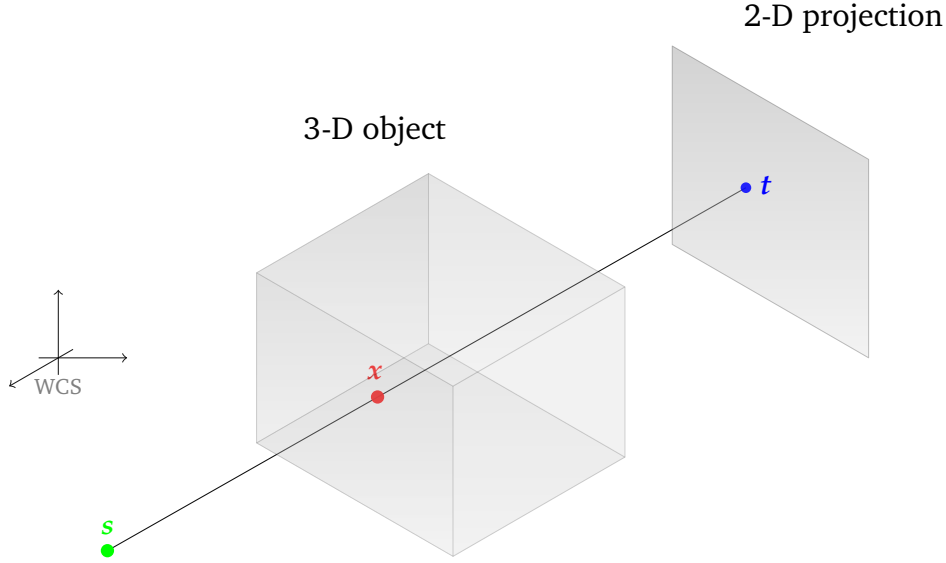


Figure 3.1: General geometry description used in this thesis. A ray is emitted at the source s targeting the detector position t while intersecting the object at position x . All three positions are described in a regular world coordinate system (WCS).

the attenuation coefficient at position x . This formulates the basis for measuring X-rays.

Rays and Attenuation

An ideal ray is defined by a source position $s \in \mathbb{R}^3$ and a target position $t \in \mathbb{R}^3$ which corresponds to the 3-D coordinate where the ray hits the detector. The line segment between s and t is described as $[st]$. Given a 3-D position $x \in \mathbb{R}^3$, the attenuation coefficient of an object at this position is described as $\mu(x) \in \mathbb{R}_0^+$. The measured intensities on the detector are described by the measurement function d_I . For a ray that is emitted from s with the intensity I_0 , the remaining respectively measured intensity $\bar{d}_I(s, t)$ at the target position t for the line segment $[st]$ is described in the mono-energetic case by the Beer–Lambert Law [Kak 88] for transmission tomography as:

$$\bar{d}_I(s, t) = I_0 \cdot e^{-\int_{x \in [st]} \mu(x) dx} \quad (3.1)$$

Detector Geometry

Since the second generation CT multiple detector elements are used for acquiring measurements. A new coordinate system on the 2-D plane of the detector represents the coordinate $\mathbf{u} = (u_1, u_2)^T \in \mathbb{R}^2$. The transform between t and the detector coordinate \mathbf{u} depends on the acquisition geometry as well as on the current projection and will be treated later. We define the total number of acquired projections P and the projection index $p \in \{1, \dots, P\}$. Each projection index p correlates to a specific source position s valid for all measurements of this projection.

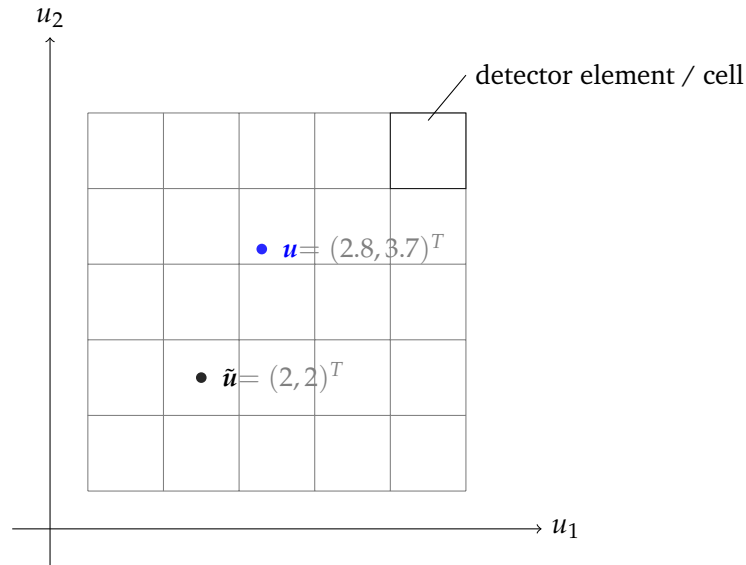


Figure 3.2: Example for a two dimensional detector grid. The discrete detector element $\tilde{\mathbf{u}}$ represents the physical measurement of a cell centered coordinate system, whereas the individual detector position \mathbf{u} needs to be approximated by interpolation.

Due to the fact that a detector acquires a fixed number of measurements respectively readings for each projection we define a reading index r and the total amount of readings R per projection. Then the 2-D layout of the detector measurements allows a one-to-one mapping between the reading index r and any integer 2-D detector position $\tilde{\mathbf{u}} = (\tilde{u}_1, \tilde{u}_2)^T \in \mathbb{N}^2$. The 2-D detector consists of R detector elements, respectively $R = U_1 \cdot U_2$, such that $\tilde{u}_1 \in \{1, \dots, U_1\}$ and $\tilde{u}_2 \in \{1, \dots, U_2\}$ as exemplarily illustrated in Figure 3.2. Here the number of detector channels is defined by U_1 . The variable U_2 represents the number of rows of the detector array.

Given the projection index p and the detector coordinates \mathbf{u} , a one-to-one correspondence – bijection – to a certain target \mathbf{t} and its source position \mathbf{s} can be established, such that the measurement function $d_I(p, \mathbf{u})$ for projection p and detector position \mathbf{u} corresponds to the measurement function of the line segment $\bar{d}_I(\mathbf{s}, \mathbf{t})$. Detector elements measure a discrete region, but not an infinitesimally small position. Therefore, a real measurement at non-discrete indices \mathbf{u} does not exist, only at discrete detector elements $\tilde{\mathbf{u}}$. This means that the acquisition data is only defined for the measurement function $d_I(p, \tilde{\mathbf{u}})$. To overcome this limitation different interpolation methods are practically used in the context of medical image processing. The easiest and computationally inexpensive one is the *nearest neighbor interpolation*. Instead of a non-discrete index, the closest discrete index is used. Alternatively, higher order interpolation methods can be used, causing higher computational costs.

The common interpolation method in the 2-D case is the *bilinear interpolation*. It computes the linear approximation of the determined value at non-discrete position

\mathbf{u} using the four surrounding detector elements. The measurement function using bilinear interpolation $\hat{d}_I(p, \mathbf{u})$ is then defined as:

$$\begin{aligned} \hat{d}_I(p, \mathbf{u}) = & (1 - \Delta u_2) \cdot ((1 - \Delta u_1) \cdot d_I(p, (\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)^T) \\ & + (\Delta u_1) \cdot d_I(p, (\lfloor u_1 \rfloor + 1, \lfloor u_2 \rfloor)^T)) \\ & + \Delta u_2 \cdot ((1 - \Delta u_1) \cdot d_I(p, (\lfloor u_1 \rfloor, \lfloor u_2 \rfloor + 1)^T) \\ & + (\Delta u_1) \cdot d_I(p, (\lfloor u_1 \rfloor + 1, \lfloor u_2 \rfloor + 1)^T)), \end{aligned} \quad (3.2)$$

with $\Delta u_1 = u_1 - \lfloor u_1 \rfloor$, $\Delta u_2 = u_2 - \lfloor u_2 \rfloor$ and the *floor* function $\lfloor \cdot \rfloor$.

More advanced interpolation methods utilize, for example, radial basis functions for interpolation. An overview and details of popular interpolation methods used in 3-D medical image processing are given by Xu *et al.* [Xu 06].

Despite the measured remaining intensity $\bar{d}_I(\mathbf{s}, \mathbf{t})$, reconstruction algorithms utilize the measured attenuation integral $\bar{d}_\mu(\mathbf{s}, \mathbf{t})$ defined as:

$$\bar{d}_\mu(\mathbf{s}, \mathbf{t}) = \int_{x \in [\mathbf{s}\mathbf{t}]} \mu(x) dx = -\ln \left(\frac{\bar{d}_I(\mathbf{s}, \mathbf{t})}{I_0} \right) \quad (3.3)$$

Note that μ identifies that the measurement function d_μ describes the remaining attenuation. Analogous to $d_I(p, \tilde{\mathbf{u}})$ we define the measurement function $d_\mu(p, \tilde{\mathbf{u}})$ by applying the I_0 -correction defined in Equation 3.3, including the bilinearly interpolated value $\hat{d}_\mu(p, \mathbf{u})$ for non-discrete detector positions \mathbf{u} . Due to the one-to-one correspondence between the reading index r and the discrete detector positions $\tilde{\mathbf{u}} \in \mathbb{N}^2$, we can define two more substitutions $d_I(p, r) \cong d_I(p, \tilde{\mathbf{u}})$ and $d_\mu(p, r) \cong d_\mu(p, \tilde{\mathbf{u}})$, such that r corresponds to $\tilde{\mathbf{u}} = (\tilde{u}_1, \tilde{u}_2)^T$.

Object

In order to reconstruct continuous objects for medical imaging, these objects are represented as discretized volumes. Such a discretized 3-D volume consists of separated volume elements also named voxels. Each voxel $\tilde{\mathbf{x}} \in \mathbb{N}^3$ is indexed in each dimension such that $\tilde{x}_1 \in \{1, \dots, X_1\}$, $\tilde{x}_2 \in \{1, \dots, X_2\}$ and $\tilde{x}_3 \in \{1, \dots, X_3\}$. While a volume consists of $J = X_1 \cdot X_2 \cdot X_3$ voxels in total, we introduce a mapped index j , such that j correlates to $\tilde{\mathbf{x}}$. For the actual size of the volume we define a voxelsize $(\delta x_1, \delta x_2, \delta x_3)^T$ in real world units. The volume size in each dimension is then given by the row vector $(X_1 \cdot \delta x_1, X_2 \cdot \delta x_2, X_3 \cdot \delta x_3)$. An example of an object discretization is illustrated in Figure 3.3.

Analogously to the discretization of the detector, the attenuation coefficients $\mu(x)$ at non-integer index positions $x \in \mathbb{R}^3$ are not represented by the discretized 3-D volume. Therefore different interpolation methods are known, using the attenuation coefficients at discrete positions $\tilde{\mathbf{x}}$. The common interpolation method in the 3-D case is the trilinear interpolation, by computing the linear approximated

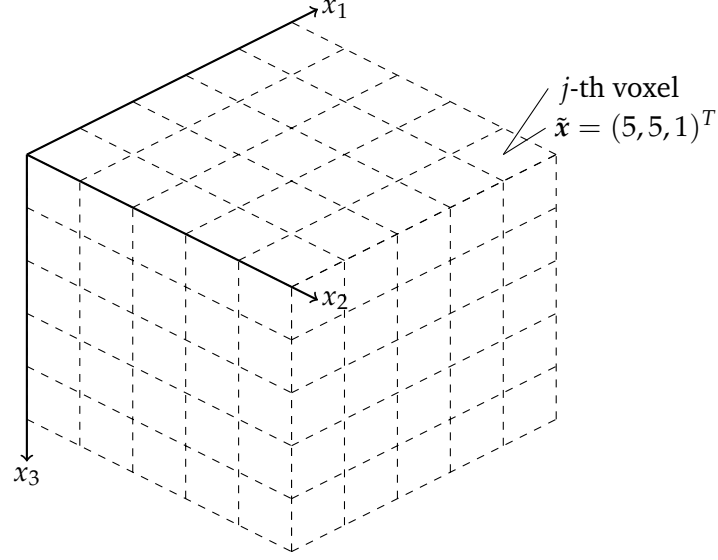


Figure 3.3: Example for an object discretization. The volume is indexed in each dimension using vector $\tilde{\mathbf{x}}$ of integer indices or equivalently indexed by j .

value between the eight voxels surrounding the non-discrete position. In this work, it is assumed that the attenuation coefficients outside of the considered volume are equal to zero. Therefore the truncation problem is not handled and we refer to the literature for details [Hopp 08, Denn 13]. The trilinear interpolation function $\hat{\mu}(\mathbf{x})$ is then defined as

$$\begin{aligned}
 \hat{\mu}(\mathbf{x}) = & (1 - \Delta x_3) \cdot \left((1 - \Delta x_2) \cdot \left((1 - \Delta x_1) \cdot \mu([\![x_1]\!] , [x_2] , [x_3])^T \right) \right. \\
 & \quad \left. + (\Delta x_1) \cdot \mu([\![x_1]\!] + 1 , [x_2] , [x_3])^T \right) \\
 & \quad + \Delta x_2 \cdot \left((1 - \Delta x_1) \cdot \mu([\![x_1]\!] , [x_2] + 1 , [x_3])^T \right) \\
 & \quad \left. + (\Delta x_1) \cdot \mu([\![x_1]\!] + 1 , [x_2] + 1 , [x_3])^T \right) \\
 & + \Delta x_3 \cdot \left((1 - \Delta x_2) \cdot \left((1 - \Delta x_1) \cdot \mu([\![x_1]\!] , [x_2] , [x_3] + 1)^T \right) \right. \\
 & \quad \left. + (\Delta x_1) \cdot \mu([\![x_1]\!] + 1 , [x_2] , [x_3] + 1)^T \right) \\
 & \quad + \Delta x_2 \cdot \left((1 - \Delta x_1) \cdot \mu([\![x_1]\!] , [x_2] + 1 , [x_3] + 1)^T \right) \\
 & \quad \left. + (\Delta x_1) \cdot \mu([\![x_1]\!] + 1 , [x_2] + 1 , [x_3] + 1)^T \right) \Big), \quad (3.4)
 \end{aligned}$$

with $\Delta x_1 = x_1 - [x_1]$, $\Delta x_2 = x_2 - [x_2]$, $\Delta x_3 = x_3 - [x_3]$.

System Matrix

In order to describe the mathematical relation between the 3-D object and the acquired 2-D projections a so called system matrix is defined. This is of importance for the theoretical background of many iterative reconstruction algorithms. Looking at the relation between the object and the projection p explains the first step of the overall concept. The discretized object description is based on the attenuation coefficients μ_j for each voxel j :

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_j \\ \vdots \\ \mu_J \end{pmatrix}, \quad (3.5)$$

where $\mu_j \cong \mu(\tilde{\mathbf{x}})$. The symbol \cong describes the bijection between both representations. The system matrix elements and its interpretation is illustrated later in Section 3.3.1. The contribution $a_{r,j}^{(p)}$ of each voxel j to a specific detector measurement $d_\mu(p, r)$ is specified by the entries of a row of the following matrix $A^{(p)}$, such that:

$$\underbrace{\begin{pmatrix} a_{1,1}^{(p)} & \dots & a_{1,J}^{(p)} \\ \vdots & & \vdots \\ & a_{r,j}^{(p)} & \\ \vdots & & \vdots \\ a_{R,1}^{(p)} & \dots & a_{R,J}^{(p)} \end{pmatrix}}_{A^{(p)}} \cdot \underbrace{\begin{pmatrix} \mu_1 \\ \vdots \\ \mu_j \\ \vdots \\ \mu_J \end{pmatrix}}_{\boldsymbol{\mu}} = \underbrace{\begin{pmatrix} d_\mu(p, 1) \\ \vdots \\ d_\mu(p, r) \\ \vdots \\ d_\mu(p, R) \end{pmatrix}}_{\mathbf{d}_\mu^{(p)}} \quad (3.6)$$

Here a single row of this matrix can be substituted by the vector $\mathbf{a}_r^{(p)}$, defined as follows:

$$\mathbf{a}_r^{(p)} = \left(a_{r,1}^{(p)} \quad \dots \quad a_{r,j}^{(p)} \quad \dots \quad a_{r,J}^{(p)} \right)^T \quad (3.7)$$

The specific system equation (Eq. 3.6) for a single projection p can also be written in short form by substitution of the partial system matrix $A^{(p)}$ as:

$$A^{(p)} \cdot \boldsymbol{\mu} = \mathbf{d}_\mu^{(p)}, \quad (3.8)$$

where $\mathbf{d}_\mu^{(p)}$ represents the measurements of projection p . In order to extend this relation for all projections P , we define the system matrix A and measurement vector \mathbf{d}_μ as:

$$A = \begin{pmatrix} A^{(1)} \\ \vdots \\ A^{(p)} \\ \vdots \\ A^{(P)} \end{pmatrix}, \quad d_\mu = \begin{pmatrix} d_\mu^{(1)} \\ \vdots \\ d_\mu^{(p)} \\ \vdots \\ d_\mu^{(P)} \end{pmatrix},$$

and state the final system of equations:

$$A \cdot \mu = d_\mu. \quad (3.9)$$

Transformations

Given the two coordinate spaces – the detector and the discretized object – we need to define the transformations between both coordinate systems. These transformations and their derivation is described by Galigekere *et al.* in detail in [Gali 03]. In interest of calculating the projected detector coordinate of a specific voxel j and a certain projection p – required for a voxel-based back-projection – we define the projection matrix B_p used for the back-projection. The projection matrix B_p for a homogeneous object index \check{x} , where

$$\check{x} = \begin{pmatrix} x \\ 1 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix}, \quad (3.10)$$

is of size 3×4 . The projection of the object index \check{x} onto the detector coordinate system can then be computed by a simple matrix multiplication and normalization of u using the inverse distance weight w^{-1} :

$$\begin{pmatrix} u_1 \cdot w \\ u_2 \cdot w \\ w \end{pmatrix} = B_p \cdot \check{x}. \quad (3.11)$$

The utilization of such projection matrices gives the flexibility for arbitrary geometries and acquisition descriptions. For ideal geometries the projection matrices can be directly derived (see [Gali 03]). For real system with non-ideal acquisitions geometries – e.g., for a C-arm system – the projection matrices are typically acquired by calibration. Details on the calibration and its advantages can be found in Strobel *et al.* [Stro 03]. Incorporating the distance weight into the projection matrix is a computational trick. Hence, we refer to the original paper by Wiesent *et al.* [Wies 00] or the description in our reconstruction benchmark [Rohk 09].

In the geometrically inverse case, the ray direction needs to be computed from the detector coordinate system for a specific projection index p . Therefore an inverse projection matrix F_p for the forward-projection is defined. This matrix is of size 3×3 and is multiplied with a homogeneous vector \check{u} that represent the detector position u , such that

$$\hat{\boldsymbol{l}} = (\boldsymbol{t} - \boldsymbol{s}) = \boldsymbol{F}_p \cdot \hat{\boldsymbol{u}} = \boldsymbol{F}_p \cdot \begin{pmatrix} u_1 \\ u_2 \\ 1 \end{pmatrix} \quad (3.12)$$

where the resulting vector $\hat{\boldsymbol{l}}$ represents the direction from the source position \boldsymbol{s} to the target \boldsymbol{t} respectively detector position \boldsymbol{u} for projection p . According to Galigekere *et al.* the inverse projection matrix \boldsymbol{F}_p as well the projection-specific source position \boldsymbol{s} can be calculated out of the projection matrix \boldsymbol{B}_p . In general the inverse projection matrix is defined by the inverse of the 3×3 sub-matrix of the projection matrix \boldsymbol{B} – by removing the fourth column.

$$\boldsymbol{F} = \begin{pmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{pmatrix}^{-1}, \quad \boldsymbol{b}_4 = \begin{pmatrix} b_{1,4} \\ b_{2,4} \\ b_{3,4} \end{pmatrix}$$

The vector \boldsymbol{b}_4 of the fourth column of \boldsymbol{B} is then utilized to calculate the projection-specific source position.

$$\boldsymbol{s} = -\boldsymbol{F} \cdot \boldsymbol{b}_4 \quad (3.13)$$

3.1.2 Reconstruction Methods Overview

To give an overview of medical image reconstruction algorithms is a quite complex task. For decades an ongoing research in this field is ever-present and year by year new or modified algorithms were introduced. Therefore, we focus on methods that are well-known, in commercial use or likely to be used in the near future. State-of-the-art research in this field, e.g., compressed sensing reconstruction methods, are not considered. For different overviews respectively grouping of reconstruction methods we refer to the literature [Kunz 07, Buzu 08, Fess 08, Zeng 10].

We divided the selected reconstruction algorithms into three different groups, while the last two groups belong to a different class of algorithms namely iterative methods. The overview is also illustrated in Figure 3.4.

The first group covers analytical reconstruction methods. The Filtered Back-Projection (FBP) [Kak 88] is a direct application of the theory of Radon integral transforms [Rado 17] in the case of the 1st generation CT providing a 2-D parallel beam geometry. The Fourier Slice Theorem (see Sect. 3.2.1) as a special case of the Radon integral transform is the basis of many analytical reconstruction methods. Therefore the FBP can be applied also for 2-D fan-beam geometry in the 2nd generation of CTs. Extending this theory then to 3-D was more complex. For the 3rd generation of CT and a helical geometry the Weighted Filtered Back-Projection was introduced by Stierstorfer *et al.* [Stie 04] and is used in practice.

For C-arm CT – where helical acquisition is not the case – a 3-D extension of the FBP method for cone-beam was introduced by Feldkamp, David and Kress commonly known as the FDK method [Feld 84]. The FDK method is used in transmission tomography for many years, since it delivers acceptable results with comparably low computational efforts.

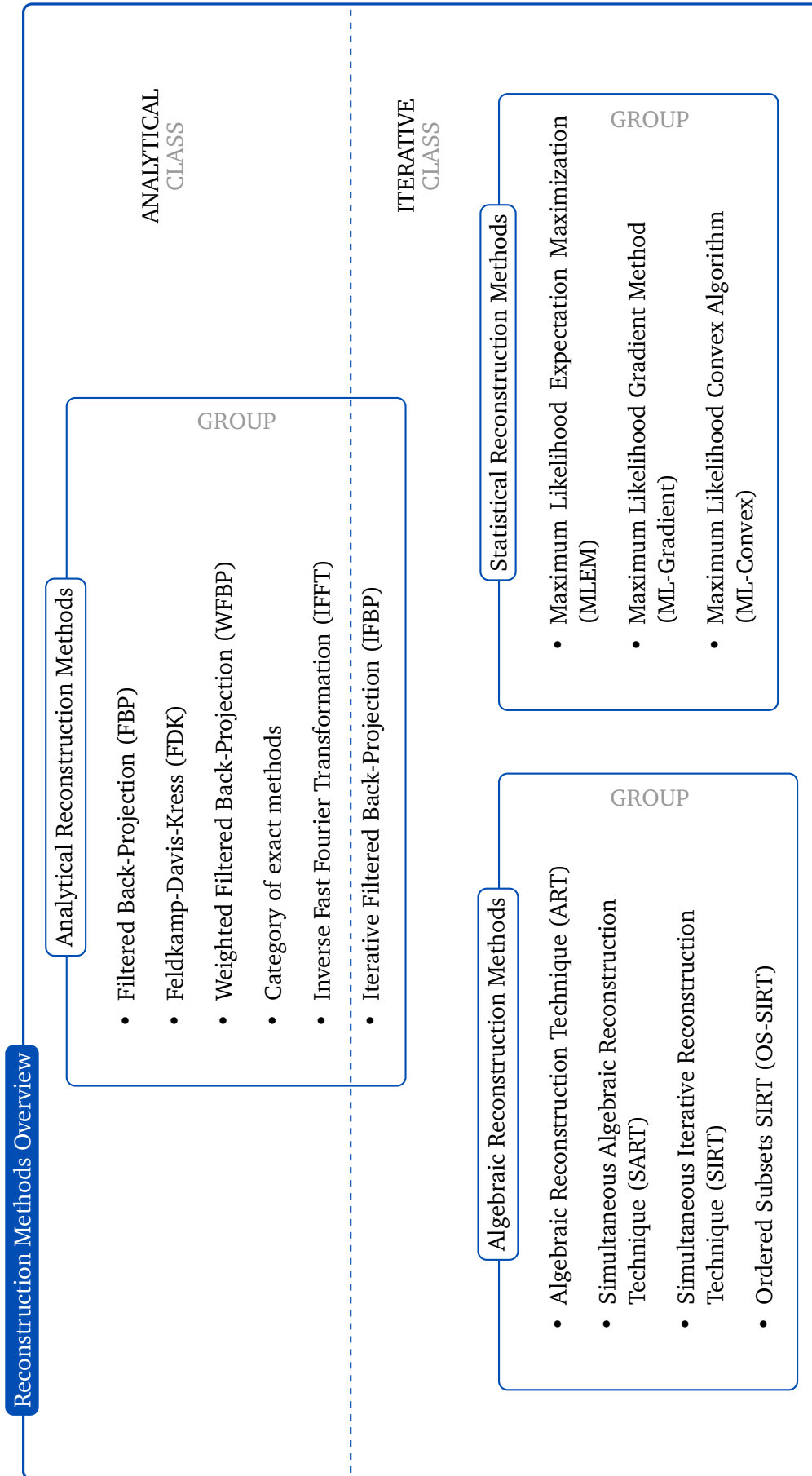


Figure 3.4: Overview of the field of reconstruction methods in medical imaging with focus on Computed Tomography.

Exact reconstruction methods – to be mentioned for completeness – form a whole new category. The analytical group is completed by reconstruction methods based on the inverse Fourier transformation [Kats 02, Denn 05].

Analytical reconstruction methods are widespread and still extensively used in many fields due to their computational performance benefits. However, they do also have some drawbacks [Maco 83]. The measurement noise generally is ignored in the problem formulation of analytical methods. In order to treat noise-related problems different kinds of pre- and post-filtering are often applied. Like the traditional Fourier Slice Theorem, analytical formulations assume continuous measurements and provide a closed form solution using continuous integrals. By discretizing these solutions, sampling issues – e.g., in Fourier space – are treated and compensated using pre- and post-processing or during interpolation on the detector. Another drawback is that analytical methods require certain standard geometries – e.g., a circular acquisition with complete sampling in radial direction and equidistant angles.

This motivates the second class of reconstruction algorithms, namely iterative methods, which can overcome these drawbacks. Also a transition to this class is given by the last method of the first group, meaning Iterative Filtered Back-Projection [Sunn 09]. We split the second class into two different groups. First the group of algebraic reconstruction methods and second the statistical reconstruction methods are introduced.

The oldest method of algebraic reconstruction is the Algebraic Reconstruction Technique (ART) [Gord 70], which was also utilized by Hounsfield for the first CT prototype. The ART was proposed by Gordon, Bender, and Herman in [Gord 70] as a method for the reconstruction of 3-D objects from X-ray photographs or electron microscopic scans. Instead of describing a continuous system and discretizing it afterwards for its application, the ART defines the problem statement as a linear algebra task. Most simplified it can be described as given a matrix A and a known vector \mathbf{d}_μ , we want to compute the unknown vector $\boldsymbol{\mu}$, such that:

$$A \cdot \boldsymbol{\mu} = \mathbf{d}_\mu \quad (3.14)$$

For invertible and moderately sized system matrices A the system can be solved easily using direct methods. However, due to the tremendous size of the system matrix in CT direct methods are not applicable.

Gordon *et al.* utilize the Kaczmarz method [Kacz 37] in order to iteratively solve the system of equations. This method will be detailed in Section 3.3.

In 1972 Gilbert *et al.* introduced the Simultaneous Iterative Reconstruction Technique (SIRT) [Gilb 72]. Instead of correcting $\boldsymbol{\mu}$ with respect to a certain row $\mathbf{a}_r^{(p)}$ of the matrix A and a single measurement $d_\mu(p, r)$, their algorithm performs a correction of $\boldsymbol{\mu}$ for all measurements \mathbf{d}_μ at a single iteration step. Therefore, at each iteration step the correction term for all measurements has to be computed and applied on $\boldsymbol{\mu}$. The performed corrections are smaller than for SART and ensure the convergence behavior. Therefore, this method provides a slower convergence.

The next improvement on algebraic reconstruction was published by Andersen and Kak in 1984 by the introduction of the Simultaneous Algebraic Reconstruction

Technique (SART) [Ande 84]. They suggested to iteratively correct μ with respect to multiple measurements $d_\mu^{(p)}$ – respectively the corresponding rows of $a_r^{(p)}$ for a single projection $A^{(p)}$ – at one iteration step. The authors argued, that this method could suppress the striping artifacts that existed in reconstructions using ART. While the Simultaneous Iterative Reconstruction Technique – published in 1972 by Gilbert *et al.* – is also able to suppress these artifacts, the SART method still provided some improvements due to a faster convergence. Hence, the convergence speed of SART in general is right between the faster ART and slower SIRT.

The idea of performing a correction with respect to a certain number of equations respectively rows of A is already used in the SART method, but can also be moved to the next step. For example instead of correcting μ for all measurements of one projection, it can be corrected for some projections n at the same time. If n equals the total amount of projections P this method becomes SIRT. Mueller *et al.* named this method Ordered Subsets Simultaneous Iterative Reconstruction Technique (OS-SIRT) [Xu 10], we used the term OS-SART [Keck 09a] instead. However, due to the history of ordered subset approaches coming from statistical reconstruction methods, OS-SIRT seems the more appropriate nomenclature. Logically the OS-SIRT method then provides a convergence speed between SART and SIRT. For further details on algebraic reconstruction methods we refer to the literature [Muel 98a].

Statistical reconstruction methods as the second group in the class of iterative methods are motivated by modelling the physics of the creation, absorption and detection of X-rays in a mathematical system. This statistical approach allows to overcome suboptimal acquisition conditions, e.g., due to a limited acquisition angle or very noisy data by low radiation dose. Limited data and very noisy data are characteristically for emission tomography, where statistical reconstruction, e.g., the Maximum Likelihood (ML) reconstruction [Shep 82] by Shepp *et al.*, is widely used. Due to the variety of statistical reconstruction methods published over the last three decades, we decided to focus on ML approaches in this work. Lange *et al.* introduced the Expectation Maximization (EM) reconstruction algorithms for emission and transmission tomography [Lang 84] in 1984. While many statistical reconstruction methods were introduced for transmission tomography over the last decades, they are still hardly used in this field due to their computational complexity. As an exceptional example we want to name the model-based iterative reconstruction (MBIR) named VEO, introduced by GE Healthcare in 2011 [GE H 11, Inte 11], which utilizes an IBM Blade Center with up to 14 Blade servers equipped with Intel Xeon CPUs. This supercomputer demonstrates the computational power required for such a product.

A complete introduction and comparison of the three named statistical reconstruction methods in Figure 3.4 can be found in the paper [Lang 95] by Lange and Fessler about globally convergent reconstruction algorithms. This paper builds the basis for our research in high performance statistical reconstruction.

This completes the overview of medical reconstruction methods in three different groups, of which selected algorithms are introduced in more detail in the following sections.

3.2 Analytical Reconstruction Methods

The class of analytical reconstruction methods represents the common class of reconstruction algorithms used in medical imaging over the last four decades. The problem to reconstruct a 2-D image from line integrals was first solved [Rado 17] by Johann Radon in 1917. Of course Radon didn't know the impact of his mathematical solution to medical imaging fifty years later.

Looking at the simplest case of reconstruction – meaning 2-D image reconstruction out of parallel beam CT like in the 1st generation CT – the problem of inverting the *Radon transform* raises. The solution of this problem is described by the Fourier Slice Theorem shortly summarized in the next subsection.

3.2.1 Fourier Slice Theorem

The Fourier Slice Theorem is a fundamental (mathematical) theorem in the field of medical imaging. Extensive text about this theorem as well as the derivation can be widely found in the literature. For example it is detailed in Chapter 6.3 of the book [Kak 88] by Kak and Slaney, as well as in the more recent book [Kram 07] (Chapter 15.4.1) by Kramme *et al.* The Fourier Slice Theorem is also well explained by Henrik Turbell in Chapter 2.1 of his dissertation [Turb 01].

The Fourier Slice Theorem can be summarized by the following main statement:

Given a parallel projection of a 2-D object, the values of the 1-D *Fourier Transform* of this projection can then be found along a radial line – in parallel to this projection – of the 2-D *Fourier Transform* of the given 2-D object.

By representing the 2-D Fourier space of the object out of the measurements of all projection angles, the reconstruction tasks leads to the inverse 2-D *Fourier Transform*. But these kind of reconstruction methods utilizing the inverse (Fast) Fourier Transformation – also called *Direct Fourier Methods* – are hardly used in practice due to interpolation inaccuracies and computational costs [Kram 07]. Instead the so called Filtered Back-Projection and its variants are the most common approach in medical imaging.

3.2.2 Filtered Back-Projection

Utilizing the Fourier Slice Theorem a reconstruction algorithm can be derived mathematically. Again we refer to the literature since we did neither introduce a continuous reconstruction space nor a continuous acquisition space. For example the derivation can be found in the dissertation [Sunn 09] on *Iterative Filtered Backprojection Methods for Helical Cone-Beam CT* by Johan Sunnegårdh (his Equation 2.5).

The Filtered Back-Projection approach is named after its implementation. Looking at the mathematical formulation the reconstruction is based on two steps. First, the projection data needs to be convolved with a high-pass kernel. Therefore this step is called filtering. And secondly, the filtered projection data has to be projected back – basically smeared along the projection direction – into the image space, the so called back-projection. Many analytical reconstruction methods share this simple design.

In an implementation we focus on a non-continuous reconstruction space μ (see Eq. 3.5) and a limited number of discretized projections d_μ (see Sect. 3.1.1). For the 2-D image reconstruction our third object dimension X_3 equals 1, as well as the number of detector rows $U_2 = 1$. We then define the filtered projection data as

$$d_g(p, \mathbf{u}) = d_\mu(p, \mathbf{u}) * g(p, \mathbf{u}), \quad (3.15)$$

where $g(p, \mathbf{u})$ denotes the filtering kernel. Analogously to $\hat{d}_\mu(p, \mathbf{u})$ we define the bilinearly interpolated version of the filtered projection data $\hat{d}_g(p, \mathbf{u})$. Here, the filter kernel is a band-limited and sampled version of the ideal – linear and spatially invariant – ramp filter. Typically this filtering is applied by multiplication of the filter kernel in the frequency domain. The frequency domain also entitles the filter name by its ramp like shape.

By modifying the filtering kernel the visual impression of the reconstructed object can be changed. E.g., with sharp edges and a higher noise level or a smoother impression with less noise. For more details, we refer to the literature and research mentioned above or [Buzu 08], since this is beyond the scope of this thesis.

For the sake of completeness, we first formulate the reconstruction algorithm of the Filtered Back-Projection treating the case of 2-D image reconstruction for the 2-D parallel beam case and look into a 3-D extension for cone-beam in the next subsection.

Using the defined geometry from Section 3.1.1 and the filtered projection data $d_g(p, \mathbf{u})$ the FBP reconstruction algorithm in the 2-D case for a half-rotation acquisition is defined as

$$\mu_j = \sum_{p=1}^P \hat{d}_g(p, u(p, j)), \quad (3.16)$$

where $u(p, j)$ represents the detector coordinate for each pixel in each projection using the projection matrix \mathbf{B}_p – see Eq. 3.11 – and the corresponding homogeneous coordinate $\check{\mathbf{x}}$, see Eq. 3.10.

Given Equation 3.16 we can formulate the instructions for a 2-D Filtered Back-Projection reconstruction:

1. Filter the acquired projection data according to Equation 3.15.
2. For each pixel j compute the reconstruction formula for μ_j in Equation 3.16.

3.2.3 The FDK method

A 3-D extension of the FBP method was published as a *Practical Cone-Beam Algorithm* [Feld 84] by Feldkamp, Davis and Kress in 1984. The method is commonly known as FDK method. This 3-D cone-beam algorithm for circular acquisition geometries is similar to the FBP method with minor differences.

In addition to the ramp filtering, the data needs to be weighted in order to compensate for both magnification and demagnification [Turb 01]. This part is called cosine weighting since the cosine of a specific angle is used. For details we refer to the literature [Denn 08] as well as Chapter 2.2.1 [Sche 11] by Scherl *et al.* This additional weighting can be easily integrated in the filtering step. In the 3-D case the number of detector rows is $U_2 > 1$, but the filtering is applied for each detector row u_2 separately. We define the filtered data including the cosine weighting for the FDK method equally to the FBP method for a fixed u_2 as

$$d_g(p, \mathbf{u}) = (d_\mu(p, \mathbf{u}) \cdot w(p, \mathbf{u})) * g(p, \mathbf{u}), \quad (3.17)$$

where $g(p, \mathbf{u})$ describes the corresponding filter and $w(p, \mathbf{u})$ the weighting for each detector coordinate \mathbf{u} of projection p . As a second difference to the FBP method the back-projection step includes an additional distance weight for each voxel at each projection. This distance weight can be incorporated into the projected detector coordinate computation using the projection matrix \mathbf{B}_p . Another difference depends on the acquisition trajectory. If instead of a full-scan trajectory only a short-scan acquisition is performed an additional Parker weighting has to be applied as part of the filtering step (see [Park 82]). The FDK reconstruction method can then be defined as

$$\mu_j = \sum_{p=1}^P w^{-2}(p, j) \cdot \hat{d}_g(p, u(p, j)), \quad (3.18)$$

where both the detector coordinate \mathbf{u} and the computation of the distance weight $w(p, j)$ are computed for each voxel in each projection. Again we formulate the instructions for a FDK reconstruction according to Equation 3.18:

1. Perform cosine weighting and ramp filtering on the acquired projection data.
2. Compute for each voxel j the defined function for μ_j from Equation 3.18.

This widely used reconstruction algorithm concludes the short introduction on analytical reconstruction methods in medical imaging. Additionally, this defines an algorithmic basis for our high performance implementation detailed in the next chapter. Before we will introduce the algebraic reconstruction methods utilized in this thesis in the following section.

3.3 Algebraic Reconstruction Methods

Instead of an analytic derivation of the reconstruction problem, the problem statement can also be formulated in a discrete mathematical formulation. These reconstruction algorithms are implied by the group of algebraic reconstruction methods as the first group in the class of iterative reconstruction methods.

In the following several algorithms are defined, which solve the mathematical formulation of the image reconstruction problem. As described in Section 3.1.1, the attenuation integrals $d_\mu(p, r)$ are acquired and represented in the measurement vector \mathbf{d}_μ . The discretized object is represented by the unknown vector $\boldsymbol{\mu}$. Given the system matrix A describing the contribution of each voxel j to each attenuation integral, the system equation is formulated as mathematical description of the reconstruction problem.

$$A \cdot \boldsymbol{\mu} = \mathbf{d}_\mu$$

In order to solve the system of equations several different algorithms can be used. Before, we shortly depict the system matrix elements.

3.3.1 Depiction of the System Matrix Elements

Each element $a_{r,j}^{(p)}$ of the system matrix describes the contribution of the voxel j to the attenuation integral $d_\mu(p, r)$. For a specific measurement, these elements are mostly zeros, except along the measurement ray from the source s to the target t for $d_\mu(p, r)$ as described in Section 3.1.1. However, the exact contribution can be interpreted differently.

In the simplest scenario, the matrix element is set to 1 if the ray passed through the voxel j and to 0 otherwise. This was proposed [Gord 70] in 1970 by Gordon *et al.*, detailed in the next subsection. Later, Shepp and Logan [Shep 74] suggested to compute the ratio between the actual intersection area or volume of the ray measured by $d_\mu(p, r)$ with the voxel j and the voxel area respectively voxel volume. A 2-D illustration is depicted in Figure 3.5 on the right side.

Another interpretation was introduced as *Fast calculation of the exact radiological*

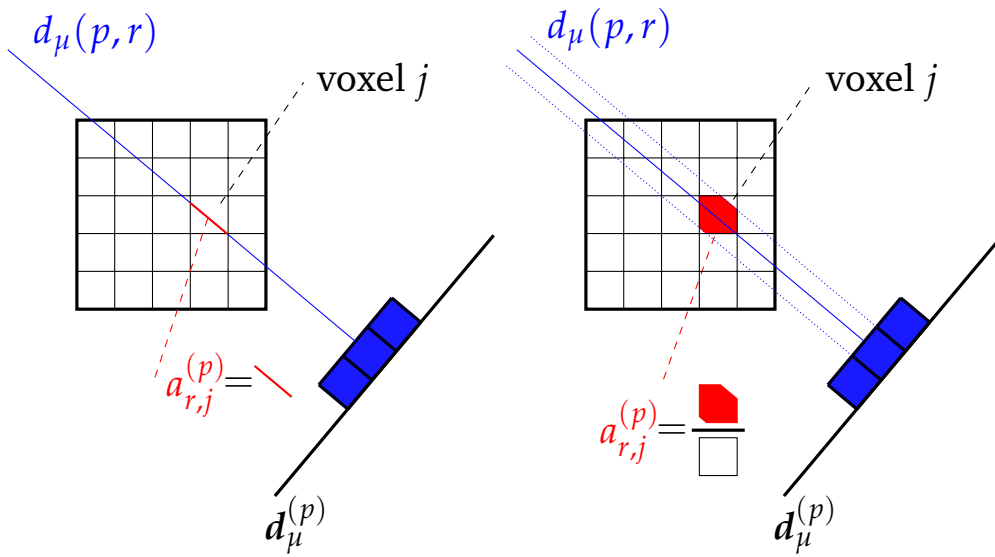


Figure 3.5: Figurative sense of the system matrix elements. On the left, the intersection length is used as system description, which was proposed by Siddon [Sidd 85]. On the right side, the intersected area is set into relation with the voxel size, proposed by Shepp and Logan [Shep 74].

path for a three-dimensional CT array [Sidd 85] by Siddon in 1985. As a system matrix element, Siddon suggested to compute the intersection length of the ray and the voxel. This is also illustrated in a 2-D example in Figure 3.5 on the left side. Since many other interpretations exist, we refer to the literature [Muel 98a, Xu 06, Kunz 07] for details and alternatives.

3.3.2 Algebraic Reconstruction Technique

The first algebraic reconstruction method was published in 1970 as an Algebraic Reconstruction Technique (ART) [Gord 70] by Gordon, Bender, and Herman. The authors suggest to compute the solution by iteratively solving the system of equations using a method published by Kaczmarz [Kacz 37] in 1937.

Due to noise in the measurement data and approximations for the system description an exact mathematical solution of Equation 3.3 typically does not exist. Therefore, the problem statement is reformulated in order to find an approximate solution to

$$\operatorname{argmin}_{\boldsymbol{\mu}} \|\mathbf{A} \cdot \boldsymbol{\mu} - \mathbf{d}_{\mu}\|_2 \quad (3.19)$$

minimizing the error. This means that the vector $\boldsymbol{\mu}$ is computed for each equation and modified towards this partial solution. Overall it is proven that this row wise correction of the unknown vector generally converges to the closest approximation of the object for a certain relaxation factor λ [Muel 98a].

We formulate the algorithmic update rule for a specific projection p and reading r as

$$\mu_j^{(\kappa+1)} = \mu_j^{(\kappa)} + \lambda \cdot \frac{d_\mu(p, r) - \sum_{c=1}^J a_{r,c}^{(p)} \cdot \mu_c^{(\kappa)}}{\sum_{c=1}^J a_{r,c}^{(p)}} \cdot a_{r,j}^{(p)}, \quad (3.20)$$

where κ represents the update index. A single iteration k is achieved by updating μ for R readings of P projections, such that

$$\mu^{(k)} = \mu^{(\kappa)}, \text{ for } k = \frac{\kappa}{P \cdot R} \text{ and } \kappa \bmod (P \cdot R) = 0 \quad (3.21)$$

The instruction for the reconstruction algorithm of the ART method can now be formulated according to the update rule:

1. First initialize $\forall j$: e.g., $\mu_j^{(0)} = 0$.
2. Iterate for a specified number of K iterations or until convergence.
3. $\forall p \in P, \forall r \in R$: update $\mu_j^{(\kappa+1)}$ for all voxels j according the update rule in Equation 3.20.

3.3.3 Simultaneous Algebraic Reconstruction Technique

The Simultaneous Algebraic Reconstruction Technique (SART) [Ande 84] represents another algebraic reconstruction methods and was introduced in 1984 by Andersen and Kak. The authors suggest a similar approach to ART, but instead of iteratively correcting μ with respect to a single row $a_r^{(p)}$ of the matrix A and a single measurement $d_\mu(p, r)$, μ should be corrected for multiple rows $A^{(p)}$ at once respectively for a single projection $d_\mu^{(p)}$ in one update step (see Eq. 3.8).

Their main argument states that this method could suppress the striping artifacts that existed in the ART reconstructions. The approximate solution is iteratively computed to the following update rule:

$$\mu_j^{(\kappa+1)} = \mu_j^{(\kappa)} + \lambda \cdot \frac{\sum_{r=1}^R \left(\frac{d_\mu(p, r) - \sum_{c=1}^J a_{r,c}^{(p)} \cdot \mu_c^{(\kappa)}}{\sum_{c=1}^J a_{r,c}^{(p)}} \right) \cdot a_{r,j}^{(p)}}{\sum_{r=1}^R a_{r,j}^{(p)}} \quad (3.22)$$

For the SART method the update index κ is incremented for P projections in each iteration, such that a single iteration is described by:

$$\mu^{(k)} = \mu^{(\kappa)}, \text{ for } k = \frac{\kappa}{P} \text{ and } \kappa \bmod P = 0 \quad (3.23)$$

The SART method is defined by the following instructions according to the update rule in Equation 3.22:

1. First initialize $\forall j: \mu_j^{(0)} = 0$.
2. Iterate for a specified number of K iterations or until convergence.
3. $\forall p \in P$: update $\mu_j^{(\kappa+1)}$ for all voxels j according the update rule in Equation 3.22.

3.3.4 Simultaneous Iterative Reconstruction Technique

Before the SART publication the Simultaneous Iterative Reconstruction Technique (SIRT) [Gilb 72] was already published in 1972 by Gilbert *et al.* The method is also able to suppress the mentioned striping artifacts of the ART method. However, compared to the SART method, the SIRT method provides a slower convergence.

The SIRT method performs a correction of μ for all measurements d_μ in a single update step. Therefore, at each iteration step the correction term for all measurements has to be computed and applied on the approximated solution vector $\mu^{(k)}$.

In order to ensure the convergence, the overall corrections of each voxel per iteration are small. This results in a slow convergence. The iteratively applied update rule of the SIRT method can be stated as:

$$\mu_j^{(k+1)} = \mu_j^{(k)} + \lambda \cdot \frac{\sum_{p=1}^P \sum_{r=1}^R \left(\frac{d_{\mu(p,r)} - \sum_{c=1}^J a_{r,c}^{(p)} \cdot \mu_c^{(k)}}{\sum_{c=1}^J a_{r,c}^{(p)}} \right) \cdot a_{r,j}^{(p)}}{\sum_{p=1}^P \sum_{r=1}^R a_{r,j}^{(p)}} \quad (3.24)$$

As a single update is applied for all readings of all projections the update index κ and iteration index k are equivalent in the case of the SIRT method. Therefore the update index κ can be replaced by the iteration index k . The instructions for the SIRT reconstruction algorithm are very similar to those of the SART method. Now the subsequent loop over all projections is brought into the update rule in Equation 3.24 such that the following steps need to be computed:

1. First initialize $\forall j: \mu_j^{(0)} = 0$.
2. Iterate for a specified number of K iterations or until convergence.
3. Update $\mu_j^{(k+1)}$ for all voxels j according the update rule in Equation 3.24.

3.3.5 Ordered Subsets for SIRT

Known from statistical reconstruction methods (see Sect. 3.4) – like the Ordered Subsets Expectation Maximization (OS-EM) [Huds 94] – another approach similar to the preceding algebraic reconstruction methods is described. Given the following order of theoretical convergence speed, the ART method provides the fastest convergence speed followed by the SART method and the slowest convergence speed using the SIRT method.

We introduce Ordered Subsets for the Simultaneous Iterative Reconstruction Technique (OS-SIRT) published in [Keck 09a]. The OS-SIRT method provides a convergence speed between SART and SIRT. While this method theoretically provides a slower convergence as the SART method, it provides computational performance benefits due to the technical realization. In parallel Xu *et al.* found equivalent results [Xu 10]. The technical realization as well as performance benefits are detailed in Section 4.2.2.

Instead of SART, where a projection-wise correction is performed, we specify ordered subsets to be used in each single update. Each ordered subset P_{OS} includes a specific number of projections. All ρ subsets then include a total of P projections. Analogously to the SART and SIRT method, we can state the update rule used in the OS-SIRT iterative algorithm:

$$\mu_j^{(\kappa+1)} = \mu_j^{(\kappa)} + \lambda \cdot \frac{\sum_{p \in P_{OS}} \sum_{r=1}^R \left(\frac{d_{\mu}(p,r) - \sum_{c=1}^J a_{r,c}^{(p)} \cdot \mu_c^{(\kappa)}}{\sum_{c=1}^J a_{r,c}^{(p)}} \right) \cdot a_{r,j}^{(p)}}{\sum_{p \in P_{OS}} \sum_{r=1}^R a_{r,j}^{(p)}} \quad (3.25)$$

For one iteration μ is updated for all ordered subsets ρ such that:

$$\mu^{(k)} = \mu^{(\kappa)}, \text{ for } k = \frac{\kappa}{\rho} \text{ and } \kappa \bmod \rho = 0 \quad (3.26)$$

The OS-SIRT method is defined similar to the SART method by the following reconstruction algorithm instructions according to the update rule in Equation 3.25:

1. First initialize $\forall j: \mu_j^{(0)} = 0$.
2. Iterate for a specified number of K iterations or until convergence.
3. $\forall \rho$: update $\mu_j^{(\kappa+1)}$ for all voxels j according the update rule in Eq. 3.25.

Given the four introduced algebraic reconstruction methods and their mathematical formulation, we conclude this section.

Another group of iterative reconstruction methods is also mathematically motivated. Taking the statistical distribution of the noisy measurements into account – e.g., by assuming Gaussian noise or Poisson statistics – the group of statistical reconstruction methods is detailed in the next section.

3.4 Statistical Reconstruction Methods

The last introduced group of reconstruction algorithms belongs to the class of iterative reconstruction methods. Statistical reconstruction methods provide different benefits, like modeling physical effects or incorporating measurement statistics. The drawback of these methods is that they are more complicated and have high computational costs. Therefore they are mostly used in applications where other methods cannot meet the clinical requirements. Statistical reconstruction methods are widely used in emission tomography and rarely in transmission tomography. Compared to transmission tomography, emission tomography is usually limited to a lower amount of measurement data which is corrupted by higher noise characteristics.

In 1982 Shepp introduced a maximum likelihood reconstruction algorithm for emission tomography, which started a significant interest in statistical reconstruction methods with applications in this field.

In this work we focus on the maximum likelihood reconstruction for transmission tomography. The presented implementation is based on the previous work by Lange and Fessler. In their paper on *Globally Convergent Algorithms for Maximum a Posteriori Transmission Tomography* [Lang 95] the authors introduce and compare maximum likelihood reconstructions by utilizing three different objective functions. They detail the widely known Expectation Maximization (EM) approach, a convex algorithm devised by De Pierro in the context of emission tomography, and a gradient-descent-algorithm introduced by the authors. We will introduce all three methods after motivating the maximum likelihood reconstruction. In this thesis we focus on the maximum likelihood convex algorithm. Therefore, this method will be derived in detail. For other statistical reconstruction methods and further details we refer to the literature [De M 09].

Before describing the statistical background of these reconstruction methods, we introduce a different notation of the forward projection, used by Lange *et al.* and for this section. The discretized version of the line integral of Equation 3.3 can also be written in inner product notation by:

$$\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle = \sum_{j=1}^J a_{r,j}^{(p)} \cdot \mu_j \quad (3.27)$$

For the statistical background, the unique pair (p, r) indicates each individual measurement for the observed $d_I(p, r)$ photons of I_0 Poisson distributed and expected photons going through the object represented by the attenuation volume $\boldsymbol{\mu}$. The Poisson nature of X-ray generation and the discretized non-overlapping volume

representation implies that the various measurements are independent and therefore follow a Poisson distribution. The intersection length for the measurement at (p, r) and the j -th volume element of $\boldsymbol{\mu}$ is defined by the system matrix element $a_{r,j}^{(p)}$. The likelihood function for the observed $d_I(p, r)$ photons is then defined as:

$$\prod_{p=1}^P \prod_{r=1}^R P(d_I(p, r) | I_0, \boldsymbol{\mu}) = \prod_{p=1}^P \prod_{r=1}^R \frac{\left(I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle} \right)^{d_I(p, r)}}{d_I(p, r)!} e^{-I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle}}, \quad (3.28)$$

where $\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle$ denotes the computation of a forward projection for the measurement ray (p, r) and the volume estimate $\boldsymbol{\mu}$. Maximizing the likelihood function is equivalent to maximizing the log-likelihood function as the log function is monotone. For many reasons it is more convenient to use the log-likelihood function, which can be derived from Equation 3.28 as:

$$\begin{aligned} L(\boldsymbol{\mu}) &= \ln \prod_{p=1}^P \prod_{r=1}^R \frac{\left(I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle} \right)^{d_I(p, r)}}{d_I(p, r)!} e^{-I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle}} \\ &= \sum_{p=1}^P \sum_{r=1}^R \left\{ \ln \left[\left(I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle} \right)^{d_I(p, r)} \right] + \left(-I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle} \right) - \ln(d_I(p, r)!) \right\} \\ &= \sum_{p=1}^P \sum_{r=1}^R \left\{ d_I(p, r) \left[\ln(I_0) - \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle \right] - I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle} - \ln(d_I(p, r)!) \right\} \\ &= \sum_{p=1}^P \sum_{r=1}^R \left\{ -I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle} - d_I(p, r) \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle + d_I(p, r) \ln(I_0) - \ln(d_I(p, r)!) \right\} \\ &= \sum_{p=1}^P \sum_{r=1}^R \left\{ -I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle} - d_I(p, r) \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle \right\} + \underbrace{\sum_{p=1}^P \sum_{r=1}^R \{ d_I(p, r) \ln(I_0) - \ln(d_I(p, r)!) \}}_h \\ &= \sum_{p=1}^P \sum_{r=1}^R \left\{ -I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle} - d_I(p, r) \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle \right\} + h, \end{aligned} \quad (3.29)$$

where h represents the substitution of the $\boldsymbol{\mu}$ -independent part of this equation. The log-likelihood results in the following equation analogous to Equation 1982 in the original paper [Lang 95], including the constant h , which can be neglected in the optimization process:

$$L(\boldsymbol{\mu}) = \sum_{p=1}^P \sum_{r=1}^R \left\{ -I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle} - d_I(p, r) \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle \right\} + h$$

By maximizing the log-likelihood we obtain the $\boldsymbol{\mu}$ which maximizes the probability for measuring the projections $d_I(p, r)$.

3.4.1 Maximum Likelihood using Expectation Maximization

In their paper, Lange and Fessler [Lang 95] first review the Expectation Maximization algorithm from the introduced maximum likelihood model. This reconstruction algorithm is derived in the paper by Lange and Carson [Lang 84]. It is explained in more details in Appendix B and is summarized in the following.

The complete data is defined as the number of photons entering $\Psi_{r,j}^{(p)}$ and leaving $\Omega_{r,j}^{(p)}$ each voxel j for each measurement. The authors then define the conditional expectations $X_{r,j}^{(p)} = E(\Psi_{r,j}^{(p)} | I_0, \boldsymbol{\mu}^{(k)})$ and $Y_{r,j}^{(p)} = E(\Omega_{r,j}^{(p)} | I_0, \boldsymbol{\mu}^{(k)})$ required for the expectation step of the EM algorithm. They further detail that both conditional expectations can be calculated by:

$$\begin{aligned} X_{r,j}^{(p)} &= d_I(p, r) + I_0 e^{-\sum_{c \in S_{r,j}^{(p)}} a_{r,c}^{(p)} \mu_c^{(k)}} - I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle}, \\ Y_{r,j}^{(p)} &= d_I(p, r) + I_0 e^{-\sum_{c \in S_{r,j}^{(p)} \cup \{j\}} a_{r,c}^{(p)} \mu_c^{(k)}} - I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle}, \end{aligned}$$

where $S_{r,j}^{(p)}$ describes the set of voxels between the source s and the j -th voxel along the measurement for reading r of projection p . Note that the j -th voxel is not included in $S_{r,j}^{(p)}$.

In order to compute the solution for the unknown $\boldsymbol{\mu}$ they suggest an algorithm by iteratively approximating the solution:

$$\mu_j^{(k+1)} = \frac{\sum_{p=1}^P \sum_{r=1}^R (X_{r,j}^{(p)} - Y_{r,j}^{(p)})}{\frac{1}{2} \sum_{p=1}^P \sum_{r=1}^R (X_{r,j}^{(p)} + Y_{r,j}^{(p)}) a_{r,j}^{(p)}} \quad (3.30)$$

The MLEM method is defined by the following instructions according to the update rule in Equation 3.30:

1. First initialize $\mu_j^{(0)} \forall j$, e.g., $\mu_j^{(0)} = \text{const.} > 0$.
2. Iterate for a specified number of K iterations or until convergence.
3. Update $\mu_j^{(k+1)}$ for all voxels j according the update rule in Equation 3.30.

3.4.2 Maximum Likelihood using Gradient-based Optimization

The second introduced algorithm by Lange and Fessler is a scaled gradient algorithm. This algorithm is more suitable for a higher performance than the MLEM algorithm, because the MLEM algorithm entails a large number of exponentiations [Lang 95]. The algorithm is motivated by heuristically maximizing the Likelihood. Therefore, the update rule for the ML-Gradient algorithm suggested in Lange *et al.* [Lang 87] corrects the unknown vector $\boldsymbol{\mu}$ by:

$$\begin{aligned} \mu_j^{(k+1)} &= \mu_j^{(k)} + \frac{\mu_j^{(k)}}{\sum_{p=1}^P \sum_{r=1}^R d_I(p,r) a_{r,j}^{(p)}} \frac{\partial}{\partial \mu_j} L(\mu_j^{(k)}) \\ &= \mu_j^{(k)} \frac{\sum_{p=1}^P \sum_{r=1}^R I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle} a_{r,j}^{(p)}}{\sum_{p=1}^P \sum_{r=1}^R d_I(p,r) a_{r,j}^{(p)}} \end{aligned} \quad (3.31)$$

This introduced algorithm provides an improvement concerning the computational cost, but comes with the lack of no guarantees for convergence of the log-likelihood function or preserving the non-negativity constraint of the unknown vector $\boldsymbol{\mu}$.

The ML-Gradient method is defined equivalent to the MLEM instructions with respect to the update rule in Equation 3.31:

1. First initialize $\mu_j^{(0)} \forall j$, e.g., $\mu_j^{(0)} = \text{const.} > 0$.
2. Iterate for a specified number of K iterations.
3. Update $\mu_j^{(k+1)}$ for all voxels j according the update rule in Equation 3.31.

3.4.3 Maximum Likelihood Convex Algorithm

Third and last statistical reconstruction method we focus on is a maximum likelihood reconstruction using a concave objective function. Lange and Fessler introduced this method in their paper [Lang 95] as convex algorithm. This is due to the fact that they rewrite the log-likelihood function by partly substitution with a strictly convex function. The negative of this function is utilized in the objective function. Therefore the objective function is concave as expected for the log-likelihood. In order to correlate this method to the convex algorithm, we refer to it as ML-Convex method in this thesis. Lange *et al.* provide proof of convergence for this method and argue that the method provides decreased computational cost compared to the MLEM method. For this algorithm we detail the derivation as this

is of importance for the implementation presented in Chapter 4.

The ML-Convex method is derived by rewriting the log-likelihood function from Equation 3.29 as:

$$L(\boldsymbol{\mu}) = - \sum_{p=1}^P \sum_{r=1}^R f_{(p,r)} \left(\left\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \right\rangle \right) \quad (3.32)$$

using the strictly convex function

$$f_{(p,r)}(t) = I_0 e^{-t} + d_I(p,r) \cdot t, \quad (3.33)$$

where I_0 and $d_I(p,r)$ state a specific measurement and t any arbitrary parameter or parameterized function. This function is convex as it represents a linear combination of two non-negative convex functions. In order to define a second objective function the authors utilize a property of convex functions. This property is also named as Jensen's inequality and therefore it is repeated here:

The Jensen inequality [Kran 99, Weis]:

For a convex function f and positive λ_i such that $\sum_{i=1}^n \lambda_i = 1$, then

$$f \left(\sum_{i=1}^n \lambda_i x_i \right) \leq \sum_{i=1}^n \lambda_i f(x_i) \quad (3.34)$$

Given the rewritten log-likelihood function $L(\boldsymbol{\mu})$, the following inequality can be derived using De Pierro [De P 93, De P 95]:

$$\begin{aligned} L(\boldsymbol{\mu}) &= - \sum_{p=1}^P \sum_{r=1}^R f_{(p,r)} \left(\sum_{j=1}^J \frac{a_{r,j}^{(p)} \mu_j^{(k)}}{\left\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \right\rangle} \frac{\mu_j}{\mu_j^{(k)}} \left\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \right\rangle \right) && \text{Jensen} \\ &\geq^* - \sum_{p=1}^P \sum_{r=1}^R \sum_{j=1}^J \frac{a_{r,j}^{(p)} \mu_j^{(k)}}{\left\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \right\rangle} f_{(p,r)} \left(\frac{\mu_j}{\mu_j^{(k)}} \left\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \right\rangle \right) \\ &= Q \left(\boldsymbol{\mu} \mid \boldsymbol{\mu}^{(k)} \right), \end{aligned} \quad (3.35)$$

where $\sum_{j=1}^J \frac{a_{r,j}^{(p)} \mu_j^{(k)}}{\left\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \right\rangle} = 1$ for all (p,r) . For the application* of the Jensen's inequality Eq. 3.34, note the negative sign in Equation 3.35 that reverses the inequality. Using this inequality the authors derive the objective function $Q \left(\boldsymbol{\mu} \mid \boldsymbol{\mu}^{(k)} \right)$ as a lower bound of $L(\boldsymbol{\mu})$. If this objective function is maximized with respect to $\boldsymbol{\mu}^{(k)}$, it is proven by this inequality that this maximizes the log-likelihood $L(\boldsymbol{\mu})$. If $\boldsymbol{\mu} = \boldsymbol{\mu}^{(k)}$ then the inequality in Equation 3.35 is an equality.

Analogously to classical EM theory the function $Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)})$ is specifically designed such that the difference $L(\boldsymbol{\mu}) - Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)})$ attains its global minimum of 0 at $\boldsymbol{\mu} = \boldsymbol{\mu}^{(k)}$. They choose $\boldsymbol{\mu}^{(k+1)}$ to maximize $Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)})$, where $Q(\boldsymbol{\mu}^{(k+1)} | \boldsymbol{\mu}^{(k)}) \geq Q(\boldsymbol{\mu}^{(k)} | \boldsymbol{\mu}^{(k)})$.

By selecting $\boldsymbol{\mu}^{(k+1)}$ for this function, the convergence can be indicated by

$$\begin{aligned} L(\boldsymbol{\mu}^{(k+1)}) &= \underbrace{L(\boldsymbol{\mu}^{(k+1)}) - Q(\boldsymbol{\mu}^{(k+1)} | \boldsymbol{\mu}^{(k)})}_0 + Q(\boldsymbol{\mu}^{(k+1)} | \boldsymbol{\mu}^{(k)}) \\ &\geq \underbrace{L(\boldsymbol{\mu}^{(k)}) - Q(\boldsymbol{\mu}^{(k)} | \boldsymbol{\mu}^{(k)})}_0 + Q(\boldsymbol{\mu}^{(k)} | \boldsymbol{\mu}^{(k)}) \\ &= L(\boldsymbol{\mu}^{(k)}) \end{aligned} \quad (3.36)$$

with strict inequality if $\boldsymbol{\mu}^{(k+1)} \neq \boldsymbol{\mu}^{(k)}$.

In order to maximize $Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)})$ instead of $L(\boldsymbol{\mu}^{(k)})$, the first derivative is taken and set to zero:

$$\begin{aligned} 0 &= \frac{\partial}{\partial \mu_j} Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)}) \quad (3.37) \\ &= - \sum_{p=1}^P \sum_{r=1}^R \frac{a_{r,j}^{(p)} \mu_j^{(k)}}{\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle} \left. \frac{\partial f_{(p,r)}(t)}{\partial t} \right|_{t = \left(\frac{\mu_j}{\mu_j^{(k)}} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle \right)} \cdot \frac{\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle}{\mu_j^{(k)}} \\ &= - \sum_{p=1}^P \sum_{r=1}^R a_{r,j}^{(p)} \cdot \left. \frac{\partial f_{(p,r)}(t)}{\partial t} \right|_{t = \left(\frac{\mu_j}{\mu_j^{(k)}} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle \right)} \\ &= - \sum_{p=1}^P \sum_{r=1}^R a_{r,j}^{(p)} \left[-I_0 e^{-\frac{\mu_j}{\mu_j^{(k)}} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle} + d_I(p,r) \right] \\ &= \sum_{p=1}^P \sum_{r=1}^R a_{r,j}^{(p)} \left[I_0 e^{-\frac{\mu_j}{\mu_j^{(k)}} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle} - d_I(p,r) \right] \end{aligned}$$

For the first derivative of $Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)})$ with respect to μ_j , the authors use the last representation:

$$\frac{\partial}{\partial \mu_j} Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)}) = \sum_{p=1}^P \sum_{r=1}^R a_{r,j}^{(p)} \left[I_0 e^{-\frac{\mu_j}{\mu_j^{(k)}} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle} - d_I(p,r) \right] \quad (3.38)$$

Equation 3.37 can be solved using Newton's method as

$$\begin{aligned}
0 &= \frac{\partial^2}{\partial \mu_j^2} Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)}) \Big|_{\mu_j = \mu_j^{(k)}} \\
&= - \sum_{p=1}^P \sum_{r=1}^R \frac{a_{r,j}^{(p)}}{\mu_j^{(k)}} \frac{\partial^2 f_{(p,r)}(t)}{\partial t^2} \Big|_{t = \left(\frac{\mu_j}{\mu_j^{(k)}} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle \right)} \\
&= - \sum_{p=1}^P \sum_{r=1}^R \frac{a_{r,j}^{(p)}}{\mu_j^{(k)}} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle I_0 e^{-\frac{\mu_j}{\mu_j^{(k)}} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle}, \tag{3.39}
\end{aligned}$$

where

$$\frac{\partial}{\partial \mu_j} Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)}) \Big|_{\mu_j = \mu_j^{(k)}} = \frac{\partial}{\partial \mu_j} L(\mu_j^{(k)}) \quad . \tag{3.40}$$

Again they use the last representation from Equation 3.39 of the second derivative of $Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)})$ such that:

$$\frac{\partial^2}{\partial \mu_j^2} Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)}) \Big|_{\mu_j = \mu_j^{(k)}} = - \sum_{p=1}^P \sum_{r=1}^R \frac{a_{r,j}^{(p)}}{\mu_j^{(k)}} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle I_0 e^{-\frac{\mu_j}{\mu_j^{(k)}} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle} \Big|_{\mu_j = \mu_j^{(k)}} \tag{3.41}$$

For $\mu_j^{(k)} > 0$ and given Equations 3.38 and 3.41 the solution of Equation 3.37 can be iteratively solved using Newton's method. This solution maximizes $Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)})$ – see Equation 3.35 – as well as the log-likelihood $L(\boldsymbol{\mu})$ in Equation 3.32. Looking at a single step of the Newton's method the iterative update rule is derived in detail:

$$\mu_j^{(k+1)} = \mu_j^{(k)} - \frac{\frac{\partial}{\partial \mu_j} Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)}) \Big|_{\mu_j = \mu_j^{(k)}}}{\frac{\partial^2}{\partial \mu_j^2} Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)}) \Big|_{\mu_j = \mu_j^{(k)}}} \tag{3.42}$$

substitute the first derivative using Equation 3.40

$$= \mu_j^{(k)} - \frac{\frac{\partial}{\partial \mu_j} L(\mu_j^{(k)})}{\frac{\partial^2}{\partial \mu_j^2} Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)}) \Big|_{\mu_j = \mu_j^{(k)}}}$$

and insert Equation 3.41.

$$= \mu_j^{(k)} - \frac{\frac{\partial}{\partial \mu_j} L(\mu_j^{(k)})}{- \sum_{p=1}^P \sum_{r=1}^R \frac{a_{r,j}^{(p)}}{\mu_j^{(k)}} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle I_0 e^{-\frac{\mu_j^{(k)}}{\mu_j^{(k)}} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle}}$$

Now exclude $\frac{1}{\mu_j^{(k)}}$ in the denominator

$$= \mu_j^{(k)} + \frac{\frac{\partial}{\partial \mu_j} L(\mu_j^{(k)})}{\frac{1}{\mu_j^{(k)}} \sum_{p=1}^P \sum_{r=1}^R a_{r,j}^{(p)} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle}}$$

and bring it into the numerator. This states the update rule – compare [Lang 95] Eq. 7 – formulated by Lange and Fessler.

$$= \mu_j^{(k)} + \frac{\mu_j^{(k)}}{\sum_{p=1}^P \sum_{r=1}^R a_{r,j}^{(p)} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle}} \frac{\partial}{\partial \mu_j} L(\mu_j^{(k)})$$

The update rule is further reformulated in order to make it applicable for our implementation. Therefore insert Equation 3.37 for the first derivate

$$= \mu_j^{(k)} + \frac{\mu_j^{(k)} \sum_{p=1}^P \sum_{r=1}^R a_{r,j}^{(p)} \left[I_0 e^{-\frac{\mu_j^{(k)}}{\mu_j^{(k)}} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle} - d_I(p, r) \right]}{\sum_{p=1}^P \sum_{r=1}^R a_{r,j}^{(p)} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle}}$$

and finally eliminate $\frac{\mu_j^{(k)}}{\mu_j^{(k)}}$. The ML-Convex update rule is then defined as

$$\mu_j^{(k+1)} = \mu_j^{(k)} + \frac{\mu_j^{(k)} \sum_{p=1}^P \sum_{r=1}^R a_{r,j}^{(p)} \left[I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle} - d_I(p, r) \right]}{\sum_{p=1}^P \sum_{r=1}^R a_{r,j}^{(p)} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle}} \quad (3.43)$$

As an alternative to the additive correction the update rule Eq. 3.43 can also be written in a multiplicative form as

$$\mu_j^{(k+1)} = \mu_j^{(k)} \cdot \frac{\sum_{p=1}^P \sum_{r=1}^R a_{r,j}^{(p)} \left[I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle} (1 + \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle) - d_I(p, r) \right]}{\sum_{p=1}^P \sum_{r=1}^R a_{r,j}^{(p)} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle}} \quad (3.44)$$

The instructions for the ML-Convex reconstruction method with respect to the update rules in Equation 3.43 and Equation 3.44 are defined as follows:

1. First initialize $\mu_j^{(0)} \forall j$, e.g., $\mu_j^{(0)} = \text{const.} > 0$.
2. Iterate for a specified number of K iterations or until convergence.
3. Update $\mu_j^{(k+1)}$ for all voxels j according the update rule in Equation 3.43 or Equation 3.44.

Iterative reconstruction methods typically include a regularization in order to incorporate prior knowledge. In the next section we shortly refer to possible constraints, which are not utilized in this work.

3.4.4 Regularization - Constraints Incorporating Priors

In order to incorporate prior knowledge or model specific physical effects, regularizations are subjoined into statistical reconstruction algorithms. These priors steer the interim solution of each iteration towards a solution with a higher probability of certain properties. E.g., high variations of the attenuation coefficients for a local neighborhood are not preferred for the reconstructed objects. Therefore, Lange and Fessler modify the introduced algorithms in order to take smoothing priors into account.

We shortly point out the idea of how to incorporate the smoothing priors introduced in the work by Lange and Fessler and refer to their paper [Lang 95] for a detailed explanation.

The authors replace the log-likelihood derived in Equation 3.29 by the log-posterior $\Delta(\boldsymbol{\mu}) = L(\boldsymbol{\mu}) - U(\boldsymbol{\mu})$, where $U(\boldsymbol{\mu})$ is some energy function penalizing large differences between neighboring pixels. The Gibbs priors introduced by Geman and McClure [Gema 85, Gema 87] take the form

$$U(\boldsymbol{\mu}) = \gamma \sum_{\{j,c\} \in O} w_{jc} \psi(\mu_j - \mu_c) \quad (3.45)$$

where γ and the weights w_{jc} are positive constants, O is a set of unordered pairs $\{j, c\}$ defining a neighborhood system and $\psi(x)$ is a potential function. The constant γ weights the overall strength of the prior. It is convenient to assume that $\psi(x)$ is even, twice continuously differentiable, and strictly convex with $\psi''(x) > 0$ for all x . De Pierro proposed an elegant alternative to Green's method [Gree 90] of handling the energy function $U(\boldsymbol{\mu})$ when maximizing $Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)}) - U(\boldsymbol{\mu})$.

Analogously to De Pierro's treatment of the log-likelihood, here the maximization of $Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)}) - U(\boldsymbol{\mu})$ is reduced to a sequence of one-dimensional maximization problems, described as $V(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)})$. Therefore it is possible to incorporate smoothing priors into a parallel implementation, which is not treated in this thesis.

3.5 Convergence and Complexity

We want to complete the third chapter with an overview of the complexity and the convergence behaviour of the considered algorithms. Before we detail the complexity calculation, we focus on the convergence behaviour.

The convergence of the analytical reconstruction algorithms is straightforward as the computation result is determined. In contrast, the convergence behaviour of iterative algorithms depends on different parameters. Most crucial is the objective function. Despite this the initialization of the object is also of importance. The better – in terms of similarity to the exact solution – the object is initialized for the iterative reconstruction the fewer iterations are required to obtain a result that is close to the ideal solution. The wide range of possible initialization strategies in this field are disregarded and we refer to the literature [Kunz 07].

As a second parameter the relaxation factor is taken into consideration. The relaxation factor is mainly used to stabilize the convergence behaviour. Smaller relaxation factors introduce a higher stability, but they also slow down the convergence. This results in a trade-off in selection of the relaxation factors for medical image reconstruction such that the reconstruction can be performed stable and efficiently.

If additional regularizations are used in the iterative process, these influence the convergence as well. The last influencing parameter is the data itself. Variations in the data also effect the convergence behaviour. Therefore, an exact convergence comparison can only be performed for the same geometry, examined object, and used parameters. In the field of medical imaging this is rarely possible and we therefore state our empirical knowledge for the considered algorithms.

In order to investigate the complexity of the considered algorithms we use the following assumptions for simplicity. Assuming a reconstruction volume with equal size in each dimension we utilize:

$$M = X_1 = X_2 = X_3 \quad (3.46)$$

Analogously to the volume we restrict each projection to a size of

$$N = U_1 = U_2 \quad (3.47)$$

such that a single projection includes $N^2 = R = U_1 \cdot U_2$ readings.

As the volume size M and the projections size N differs in each dimension only in a linear scaling factor, we note that $N = \alpha \cdot M$. In the following we assume that the volume consists of N^3 elements. Looking at the back-projection operator N^3 elements have to be incremented for each projection. Then the complexity of the (filtered) back-projection for a total of P projections is $\mathcal{O}(PN^3)$.

For iterative reconstruction methods the forward-projection operator has to be additionally considered. Forward-projection algorithms vary in their complexity. Differing examples can be found in literature [Xu 06] as well as in the dissertation

	(F)BP	ART	SART	OS-SIRT	SIRT	ML-Convex
complexity	$\mathcal{O}(PN^3)$	$\mathcal{O}(KPN^3)$	$\mathcal{O}(KPN^3)$	$\mathcal{O}(KPN^3)$	$\mathcal{O}(KPN^3)$	$\mathcal{O}(KPN^3)$
iterations*	1	3 – 5	5 – 10	10 – 20	> 20	> 30
volume updates	1	$K \cdot P \cdot R$	$K \cdot P$	$K \cdot \rho$	K	K

Table 3.1: The complexity of analytical and iterative reconstruction algorithms is compared for the simplified case. Here each dimension of the reconstructed volume as well as each dimension of the utilized projections are of size N . The total amount of projections P is utilized for a total of K iterations. In the case of OS-SIRT the projections are split into ρ ordered subsets. The convergence behaviour decreases from left to right. Analogously, the suggested number of iterations* increases. In order to depict another important parameter, the number of volume updates, is also compared.

(p.43) [Kunz 07] by Holger Kunze. For our implementation we need to approximate the forward-projection complexity. In case of a ray casting we assume N^2 rays sampled at N volume elements. For P projections this results in a total complexity of $\mathcal{O}(PNN^2) = \mathcal{O}(PN^3)$.

Combining both operators in iterative algorithms results in a complexity of $\mathcal{O}(PN^3)$ for each iteration. For K iterations the complexity can then be estimated as $\mathcal{O}(KPN^3)$.

Concluding Chapter 3 on Medical Image Reconstruction Algorithms an overview of the complexities, the required volume updates and the proposed number of iterations as an indicator for the convergence is stated in Table 3.1.

3.6 Conclusion

In order to revise the utilized medical image reconstruction algorithms the nomenclature used in this thesis is introduced. It includes definitions of the geometry, acquisition data, reconstruction object, system matrix, and the required transformations. An overview of the diversity of medical reconstruction algorithms and a possible categorization is given. This three different categories of reconstruction methods – namely analytical, algebraic and statistical reconstruction methods – are further explained by several examples for each group. While the focus of this thesis is on the class of iterative reconstruction algorithms, examples for the widely used analytical reconstruction methods are introduced, too. This eases the comparison of the described algorithms in terms of algorithmic complexity and computational differences. The different introduced algorithms of each group build the basis of the medical image reconstruction methods used in our research. Finally, a comparison of those highlights their important differences in terms of convergence and algorithmic complexity.

High Performance Medical Image Reconstruction

Given the technical and theoretical background of our research in Chapter 2 and Chapter 3 the main scientific contribution in the field of high performance medical image reconstruction is shown in this chapter.

In medical image reconstruction as well as medical image processing complicated algorithms have to be computed with very strict time constraints. These time constraints typically depend on the particular application. For example, in interventional radiology delays are critical while surgeons operate. In case of emergency medicine the patient treatment during the golden hour is vital [Nico08]. For screening methods examination quality together with high patient throughput builds the economical emphasis. In order to meet these time constraints vendors draw on a variety of high performance computing solutions such as custom hardware like field-programmable gate arrays (FPGAs), highly optimized CPU implementations or nowadays graphics cards. While all these platforms offer plenty of parallel computing units, each has its own programming language and tool chain, e.g., VHDL for FPGAs, SIMD intrinsics for CPUs, OpenGL or CUDA for GPUs.

In this work we accelerate dedicated operations of selected reconstruction algorithms using the compute power of GPUs in combination with NVIDIA's CUDA. The first section details possible implementations, optimizations and important facts for the parallelization of the utilized operators as well as available numerical libraries. Therefore each operator and its modification is detailed as well as evaluated with respect to computational performance. The exemplary comparison shows the fastest measurement out of several if not stated differently.

In Section 4.2 the realization scheme for each reconstruction algorithm is illustrated. Here different compositions of the introduced operators with certain extensions and specifically added operations manifest the scientific contribution of our work.

The chapter is concluded with a performance comparison between CUDA and the alternative OpenCL (see Sect. 2.4.1). The performance of both APIs is stated for the introduced two operators – namely forward- and back-projection.

4.1 High Performance CUDA Implementations

The Compute Unified Device Architecture offers a unified hardware and software solution for parallel programming and computation on NVIDIA GPUs supporting the standard C programming language (see Chapter 2). The difficulty in programming GPUs using CUDA on one side is the problem parallelization as the algorithm has to be divided into many – at best independent – sub-algorithms with similar computations. The challenge on the other side is then to understand the underlying hardware and modify or select a parallelization scheme such that high computation rates respectively high performance can be achieved.

Starting with a numerical library example for the Fast Fourier Transformation, the acceleration for pre- and post-processing algorithms is exemplified. Afterwards our implementations of the focused operators – back- and forward-projection – are introduced.

4.1.1 Fast Fourier Transformation

As part of their hardware and software development for CUDA, NVIDIA also introduced GPU accelerated libraries. Part of those libraries are highly optimized versions of commonly used algorithms such as matrix multiplication or the Fast Fourier Transformation. As such algorithms are suitable for parallelizations they are typically used to prove the achieved peak performance compute rates for the underlying hardware.

In case of the FFT NVIDIA published the cufft-library [NVID07b] offering accelerated parallel computations of the discrete Fourier transforms for complex or real-valued datasets. As the FFT is used twice in the filtering step of Section 4.2.1, we reference the utilized library without further details. In this field we would like to refer to the research of Vasily Volkov:

Volkov *et al.* did not only publish the research results on FFT implementations for CUDA GPUs [Volk08b] – which was 2.9 times faster than the available cufft-library 1.1 – the results were also incorporated into newer versions of NVIDIAs numerical libraries. One of the latest results on the parallel FFT [Volk10] shows a contradiction to NVIDIAs programming advice of trying to achieve a high multiprocessor warp occupancy as better performance could be achieved at lower occupancy with higher workload.

algorithm	smoothing filter	Laplacian decomp.	Laplacian comp.	multiscale filter	DMA CPU↔GPU	overall
CPU 1-core	181.59	18.38	15.99	115.35		331.31
CPU 2-cores	91.57	9.88	7.91	58.94		168.30
GPU	3.90	0.86	0.48	0.74	1.26	7.24
speedup	46.6	21.4	33.3	155.9		45.8

Table 4.1: The measured computation times for each filtering step are given in milliseconds for optimized programs on a single-core, a dual-core Intel Xeon 5150 CPU (2.66GHz) as well as for NVIDIAs GeForce 8800 GTX GPU. On the bottom the achieved GPU acceleration is stated in comparison of the GPU and the single-core CPU.

4.1.2 Pre- and Post-Processing

The parallel compute power of GPUs is well suited for pre- and post-processing algorithms in Medical Image Processing. As part of our research a real-time multi-scale time- and motion-dependent noise-reduction algorithm for angiographic projections was realized using CUDA [Buhr 09]. Due to the fact that this filter is typically applied during acquisition and not incorporated into the proposed reconstruction algorithms, only a short introduction as well as the achieved performance is stated.

Real data measurements come along with the problem of noise. Therefore, noise reduction methods are mandatory in medical imaging and can be applied in pre-processing as well as post-processing steps. As an example for the GPU acceleration in the area of pre- and post-processing we realized a compute-intensive filter for real-time structure-preserving reduction of pixel noise in angiographic projection images. The filter is composed of different algorithms. A single projection – in this example of size 960^2 pixels – is decomposed into different levels using the laplacian pyramid [Burt 83]. Each level is then smoothed by an edge-preserving directional variance-driven filter in a local neighborhood. Afterwards the multi-scale filter incorporates the pixel information from previous filtered projections depending on the correlation for each level separately. As last step the final image results out of the Laplacian composition.

The achieved performance measurements for a single CPU core, two CPU cores and the GPU using CUDA for each filtering step as well as the data transfer towards and from GPU memory are stated in Table 4.1. Given an overall performance of 7.24 ms per image for the GPU computation enables realtime processing of angiographic projections. With an overall acceleration factor of 45.8 compared to a single CPU core, this sets an encouraging example of GPUs parallel compute power using CUDA. For further details we refer to our published research results [Buhr 09].

4.1.3 Back-Projection

One of the main contributions of this thesis is the back-projection operator which is similarly used in almost every medical image reconstruction algorithm. Therefore the performance, flexibility and usability of this operator is crucial for high performance medical image reconstruction.

In 2007 we presented a high performance back-projection [Sche 07b] as one of the earliest approaches for GPU-based medical image reconstruction using CUDA. Looking at the back-projection algorithm two different strategies can be implemented. The Algebraic Reconstruction Technique proposed by Gordon *et al.* (see Sect. 3.3.2) computes a ray-driven back-projection, where each affected voxel is increased by the appropriate value (see Fig. 3.5). Alternatively the back-projection can be implemented as a voxel-driven approach which is most common [Buzu 08] and our choice in the proposed implementations.

Algorithm 4.1: Back-projection of the p -th projection image.

```

input : volume  $\mu$ , projection-matrix  $B_p$ , measurements  $d_\mu^{(p)}$ 
output: updated  $\mu$ 

// for each voxel  $\tilde{x}$  resp.  $j$  in the volume  $\mu$ 
for  $\tilde{x}_3 = 1$  to  $X_3$  do
  for  $\tilde{x}_2 = 1$  to  $X_2$  do
    for  $\tilde{x}_1 = 1$  to  $X_1$  do
      // compute  $\mathbf{u}$  and  $w$  see Eq. 3.11
       $\begin{pmatrix} w \cdot \mathbf{u} \\ w \end{pmatrix} \cong B_p \cdot \tilde{\mathbf{x}};$ 
      // update the volume element see Eq. 3.18
       $\mu_j = \mu_j + \hat{d}_\mu(p, \mathbf{u});$ 
    end
  end
end

```

We think that in this context a ray-driven back-projection is less suitable for parallelization due to possible race conditions for the parallel voxel data access. Using CUDA these race conditions can be prevented using *atomic functions*. However this usage also reduces the performance significantly. Furthermore, these approaches tend to introduce high-frequency artifacts that manifest as Moiré patterns in the final reconstruction result [De M 02, De M 04].

Instead of the ray-driven back-projection the voxel-driven approach simulates a ray through the center of each voxel hitting the detector at a non-discrete position \mathbf{u} . Using the bilinear interpolated measurement $\hat{d}_\mu(p, \mathbf{u})$ at the position \mathbf{u} each voxel j can be updated independently. This results in an ideal parallel problem as the measurement values for each projection are accessed read-only.

Looking at the voxel-driven back-projection of a single projection the algorithm is stated in Algorithm 4.1. Given a volume μ that is defined at the whole-numbered indices \tilde{x} , the measurement vector $\mathbf{d}_\mu^{(p)}$ for projection p and the transformation given by the projection matrix \mathbf{B}_p each volume element is updated with the bilinear interpolated detector value at the corresponding position \mathbf{u} . Both, \mathbf{u} and w are computed using the matrix-vector product $\mathbf{B}_p \cdot \tilde{x}$ of the projection matrix and the homogeneous volume coordinate (see Eq. 3.11). Note that if a weighting w is applied, – e.g., in case of the FDK algorithm – then \mathbf{u} is computed by normalization using w^{-1} and the increment is weighted before updating μ_j . If no weighting is applied, the weight is defined as $w = 1$ and can be ignored. In Algorithm 4.1 no weighting is applied as this will be detailed for each reconstruction algorithm separately in Section 4.2.

Implementation

Given NVIDIA's grid- and block-layout for the parallelization of an algorithm using CUDA, different parallelization schemes are possible. A naive approach is the parallelization over all target elements. Meaning each CUDA thread computes a single back-projection for one-voxel such that all inner loops over x_1, x_2, x_3 are realized using this parallelization scheme (see Sect. 2.3.1). As this has a major performance drawback, we propose a different scheme in the following. The comparison for both approaches is evaluated in Section 4.3.

The example code for the voxel-driven CUDA implementation is shown in the listing of Figure 4.2. Our parallelization scheme suggests a partitioning over the x_1, x_3 axes as illustrated in Figure 4.1. The x_2 -axis is then executed as innermost loop for each kernel (see line 23) providing the following two benefits. Key advantage of this scheme is that neighboring threads in first dimension always access the volume data coalesced. This coalesced memory access is necessary in order to maximize the achieved memory bandwidth (see Sect. 2.3.5). As a second benefit the major part of the geometric computation – matrix-vector product – can be further reduced as only x_2 is varying in the for-loop of the kernel. This allows the implementation of an incremental version of the back-projection kernel reducing the required number of instructions inside the innermost loop (see line 19 – 21 and 26 – 28). Saving six multiply-add operations by incrementing the homogeneous coordinates with the appropriate column of \mathbf{B}_p for neighboring voxels in x_2 -direction represents a reduction of $\sim 39\%$ of the innermost loop instructions.[Sche 11, p.81]

Accessing the discrete measurement data at arbitrary positions requires interpolation as discussed in Section 3.1.1. As GPUs provide specific texture units the interpolation scheme for a bilinear interpolation of the measurement data $\hat{\mathbf{d}}_\mu(p, \mathbf{u})$ is preferable. Such an implementation using the texture units can provide additional performance benefits [Schw 11]. For each back-projection the measurement data is copied into a 2-D texture array and accessed via the CUDA texture fetching function `tex2DO` (see line 37). This enables the implicit bilinear interpolation by the texture units as well as automatic usage of the existing texture cache.

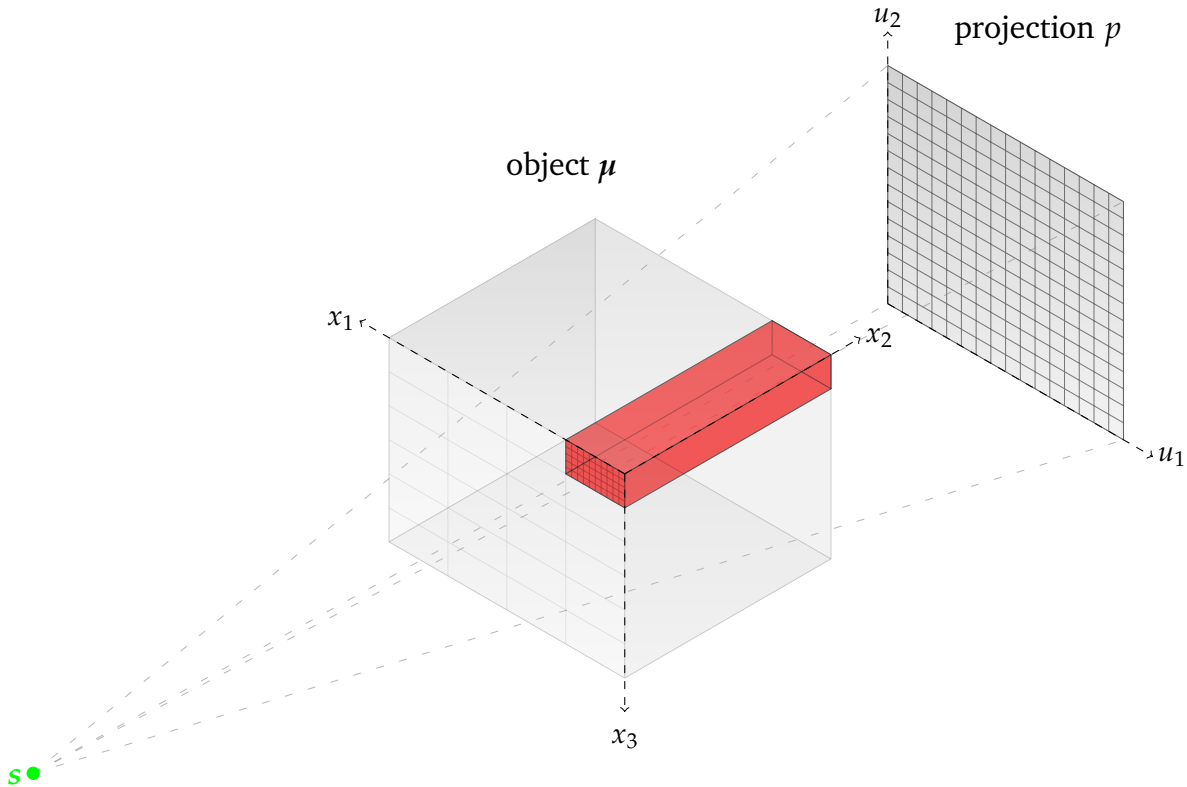


Figure 4.1: Back-projection geometry and suggested parallelization scheme. The x_1 - x_3 -plane illustrates the partitioning into the CUDA grid and a CUDA thread-block (colored in red). Each parallel thread then computes all updates along the x_2 -axis.

Using textures for the measurement data also introduces further benefits. If the computed texture coordinate \mathbf{u} is out of the defined size, no measurement data exists. In this case the computed coordinate has to be checked before each texture access and dismissed if not defined. Setting the specific *addressMode* – defining how out-of-range texture coordinates are handled – to *cudaAddressModeClamp* out-of-range texture coordinates are clamped to the valid range. Enlarging the measurement data by one element at each edge and setting each element’s value to zero, totally avoids any coordinate checks for out-of-range texture accesses.

If the measurement data at the edges contains values greater than zero, this implementation trick has a minor drawback of introducing possible errors. The texture units will then compute the bilinear interpolated value between zero and the edge values for positions \mathbf{u} directly at the edges with a lower measurement value than might be expected. As the measurement data is truncated in such case, different possibilities exist which are not discussed here.

The dimensions of texture coordinates for GPU programming are defined from $[0, \dots, \mathbf{U}]$. Each measurement value is defined at .5 positions. In order to compensate the 0.5-offset this can be easily included inside the texture fetching function as $\text{tex2D}(\text{texture}, u1 + .5f, u2 + .5f)$. Though this introduces two additional add operations for each texture fetch, which can be smartly saved. The out-of-range texture

clamping introduces another offset of 1.0 for the position \mathbf{u} . Both, out-of-range texture clamping offset and the 0.5 coordinate offset for texture access are incorporated into the homogeneous part of the projection matrix \mathbf{B}_p and thus included into the geometric computation without any additional operations (see line 19-21).

The last implementation detail describes the access of the projection matrix \mathbf{B}_p for the geometric computation. Each parallel thread has to access the projection matrix several times. Looking at Table 2.1 two possible implementations with high performance are suitable as a low register usage is mandatory. The obvious method uses constant memory (see line 1) in order to store twelve elements of the projection matrix in a globally defined array. The elements are copied into constant memory using the CUDA function *cudaMemcpyToSymbol()* before kernel execution. Inside the kernel the matrix elements then can be accessed by the defined array variable. We found that the usage of shared memory instead can provide minimal performance benefits. Here the projection matrix is copied into global memory before kernel execution using the CUDA function *cudaMemcpy* with the attribute *cudaMemcpyHostToDevice*. At the very beginning of the kernel execution the first twelve threads of each CUDA block read one element of the projection matrix and store it into the defined shared memory. Before the kernel continues with the common implementation, all threads of each block have to be synchronized using the CUDA function *__syncthreads()* which acts as a barrier as described in Section 2.3.2. Similar to the constant memory the matrix elements stored in shared memory can be accessed through the array variable of the declared shared memory. At specific instruction addresses each thread reads the same matrix element from shared memory. This broadcast access reduces bank conflicts for the shared memory access [NVID 10a].

```

1  __constant__ float kpBp[3*4];    // constant memory
2
3  texture <float, 2, cudaReadModeElementType> Texture;
4
5  __global__ void kernelBackProjectIncremental(
6      float* dpVolume,
7      const int X2,
8      const int volumeStrideX1,
9      const int volumeStrideX2)
10 {
11     // compute volume index x1 and x3
12     int x1 = __umul24(blockIdx.x, blockDim.x) + threadIdx.x;
13     int x3 = __umul24(blockIdx.y, blockDim.y) + threadIdx.y;
14
15     // compute memory index to first voxel in x2 column
16     int idx = __umul24(x3, volumeStrideX2) + x1;
17
18     // compute static part of matrix vector product
19     float u1tmp = kpBp[0] * x1 + kpBp[6] * x3 + kpBp[9] ;
20     float u2tmp = kpBp[1] * x1 + kpBp[7] * x3 + kpBp[10];
21     float wtmp  = kpBp[2] * x1 + kpBp[8] * x3 + kpBp[11];
22
23     for (int x2=0; x2 < X2; x2++, idx += volumeStrideX1)
24     {
25         // compute incremental matrix vector product
26         float u1 = u1tmp + kpBp[3] * x2;
27         float u2 = u2tmp + kpBp[4] * x2;
28         float w  = wtmp  + kpBp[5] * x2;
29
30         // compute projection coordinates
31         float norm = 1.0f / w;
32         u1 = u1 * norm;
33         u2 = u2 * norm;
34
35         // compute weight if utilized
36         float weight = norm*norm;
37
38         // fetch measured value
39         float val = tex2D(Texture, u1, u2);
40
41         //increment voxel value
42         dpVolume[idx] += weight * val;
43     }
44     return;
45 } // kernelBackProjectIncremental

```

Figure 4.2: Back-projection kernel code for NVIDIA GPU using CUDA. Loop over all elements of one line in x_2 -direction, parallelization over the x_1 - x_3 -plane.

Performance

In order to exemplarily evaluate the achieved performance we used a dataset from an angiographic C-arm system (see Fig. 1.4). The dataset consists of 414 projections of 1024^2 pixels each. For a volume consisting of 512^3 voxels, we have chosen a voxel size of 0.26^3 mm^3 in order to ensure that all voxels reside inside the field-of-view. Further on we refer to this dataset as DataSet A (see Sect. A.1).

The corresponding kernel program of our CUDA implementation results in a total register usage of only 10 registers. This is owed not least to the incremental approach that also reduces the register usage. Looking at the CUDA occupancy calculator, we could achieve the maximal multiprocessor warp occupancy for the G80 GPU generation, which was mandatory to hide memory latencies. By implementing the back-projection in the innermost loop for two projections instead of a single projection we achieved a register usage as low as 11 registers. As this is not optimal the resulting occupancy and performance achievement on the G80 is lower and therefore not discussed.

For the CUDA programming model the grid- and block-configuration influences both the global memory access pattern and the texture cache usage. Therefore it is recommended to evaluate the performance for different configurations separately. Table 4.3 clearly indicates the performance differences for the selected grid- and block-configurations. In our experience the configuration of a block-size of 128×2 threads was optimal for most reconstructions and therefore used in our evaluation results.

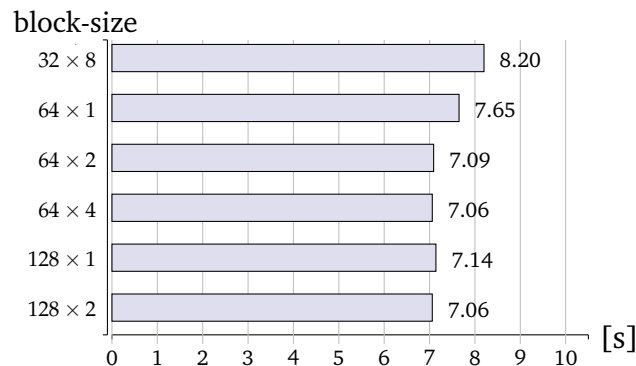


Figure 4.3: Exemplary back-projection performance on the GeForce 8800 GTX for different grid- and block-configurations. The overall back-projection time of utilized evaluation dataset is stated in seconds for the different block-sizes. Out of multiple runs the fastest achieved time is stated.

Before NVIDIA introduced the GeForce 8800 GTX GPU together with the CUDA framework, the highest single chip computational performance in the field of medical image reconstruction was achieved using the IBM CELL processor. Therefore we compared a single CELL processor with the GeForce 8800 GTX on back-projection performance. The GPU could achieve almost a factor of 3 times better performance than the CELL processor. Here the bilinear interpolation is weighty on the resulting performance as the CELL processor lacks for texture units. Without bilinear interpolation, instead using nearest-neighbor interpolation the performance benefits of the

GPU is narrowed to a factor of 1.6, which is almost exact the ratio of the theoretical peak performance of both devices.

As this holds also for the CPU, we give another example for the reconstruction performance. Here the exemplary angiographic C-arm dataset is slightly different using 543 projections each 1240×960 pixels (see Sect. A.2). For comparison an Intel Core2Duo Conroe CPU @2.4 GHz (E6600) is compared with the NVIDIA QuadroFX 5600 GPU. For optimization the CPU implementation was vectorized using SSE intrinsics as well as multi-threading. The achieved results of both highly optimized CPU and GPU implementations are stated in Table 4.2. Without the cost of bilinear interpolation the GPU can perform the total of 543 back-projections up to 16.9 times faster than the dual-core CPU. Including the bilinear interpolation into the CPU-GPU comparison again state the massive parallel compute power of GPUs.

BP performance	CPU	GPU	speedup
bilinear interpolation	362 s	12.4 s	29.2
nearest-neighbor	210 s	12.4 s	16.9

Table 4.2: Back-projection comparison example for an Intel Core2Duo Conroe CPU (E6600) with 2 cores (2.4 GHz) and a NVIDIA QuadroFX 5600 GPU. For the measurement 543 projections each 1240×960 pixels were back-projected onto a 512^3 volume. The measurement includes both times for back-projection with bilinear and nearest-neighbor interpolation.

Bandwidth limitation

The massive parallelism of GPUs requires a much higher memory bandwidth compared to CPUs (see Fig. 2.2). We show that our proposed CUDA implementation is bandwidth limited [Sche 10], even given a memory bandwidth of 86.4 GB/s for the GeForce 8800 GTX.

In Table 4.3 three different GPUs from the G80 generation are compared. Looking at the shader clock frequencies the compute performance increases from left to right. The GeForce 8800 GTX runs at 1350 MHz. This is slightly increased by 50 MHz for the QuadroFX 5600 to 1400 MHz. With 1.5 GHz the highest shader clock frequencies are given by the GeForce 8800 Ultra graphics card. Looking at the back-projection performance some would expect that with increasing shader clock frequency the performance increases, respectively measured computation time decreases. In case of comparing the computation time for both GeForce GPUs this seems correct. But looking at the NVIDIA QuadroFX 5600 the opposite happens. With additional 50 MHz of shader clock frequency the computational time is almost one second slower. Focussing at the memory bandwidth of the QuadroFX it is noticed that the bandwidth decreased compared to both opponents. This indicates that our implementation is limited by memory bandwidth and can be further supported by the following simple calculation:

$$7.19\text{s} \cdot \frac{900\text{MHz}}{800\text{MHz}} = 8.09\text{s} \quad (4.1)$$

Taken the 7.19s computation time and scale it by the ratio of both memory frequencies, the expected computation for the QuadroFX 5600 of 8.09 seconds is almost achieved with an average of 8.14 seconds. Repeating this theoretical experiment, the NVIDIA GeForce 8800 Ultra graphics card should achieve a computation time of 5.99 seconds. We experimentally measured a slightly faster computation time of 5.93 seconds. This indicates that other parameters like the shader clock marginally influence the overall performance. For the proposed CUDA implementation the highest influence is given by memory bandwidth of the utilized graphics cards. This concludes the implementation of the back-projector operator, looking at the forward projection next.

graphics cards	GeForce 8800 GTX	QuadroFX 5600	GeForce 8800 Ultra
chip clock	575 MHz	600 MHz	612 MHz
shader clock	1350 MHz	1400 MHz	1500 MHz
db. memory cl.	900 MHz	800 MHz	1080 MHz
memory BW	86.4 GB/s	76.8 GB/s	103.7 GB/s
BP performance	7.19 s	8.14 s	5.93 s

Table 4.3: The properties of three different G80 chips are listed. For the performance measurement 414 projections each 1024^2 pixels were back-projected onto a 512^3 volume (see Sect. A.1).

4.1.4 Forward-Projection

The second main contribution of this thesis is the forward-projection operator which is required in addition to the back-projection operator for all iterative reconstruction algorithms. In this case, the implementation's performance, flexibility as well as accuracy are of high impact for high performance iterative medical image reconstruction.

In 2008, we presented the first comparison of high-speed forward-projection implementations on GPUs utilizing OpenGL and CUDA [Wein08]. The presented ray casting approach was realized using the OpenGL as well as the CUDA 1.1 and CUDA 2.0 APIs. Before detailing our implementations, optimization and evaluation results, the preference to the ray casting approach as forward-projection operator is discussed.

Iterative medical image reconstruction approaches utilize the forward-projection operator for the simulation of the Beer-Lambert law (see Eq. 3.1) using a discrete object respectively the estimated reconstruction result. In Chapter 3 this forward-projection was mathematically defined for both algebraic and statistical reconstruction methods using the system matrix (see Eq. 3.8). Therefore the continuous line-integral of ray r for projection p is approximated with a discrete sum given by:

$$\int_{x \in r} \mu(x) dx \approx \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu} \rangle = \sum_{j=1}^J a_{r,j}^{(p)} \cdot \mu_j \quad . \quad (4.2)$$

Depending on the depiction of the system matrix elements (see Sect. 3.3.1) this results in different operators, e.g., the Siddon method [Sidd85]. As many other popular methods exist, e.g., the Joseph method [Jose82], we decided to use the ray casting method based on the following arguments. The ray casting approach was first published and used in the field of computer graphics. There it is used for several applications like the computation of object intersections as well as for volume rendering. Since the graphics processing unit is optimized for computer graphics in the first place, the ray casting approach is very suitable for a GPU implementation. Siddon's method needs to compute the intersection length for each ray with each intersected voxel. This drawback results in a high computational effort since these values cannot be stored efficiently. The ray casting approach instead provides a more simpler solution that is easy to parallelize.

For iterative reconstruction approaches the accuracy of the forward-projection operator is important. Therefore, we refer to the work of Xu *et al.* on a comparative study of popular interpolation and integration methods for use in computed tomography [Xu06]. Xu *et al.* uses a different notation for many integration strategies and therefore is difficult to correlate to this thesis. The important ones are stated here in order to optionally prepare the reader for the paper. The *slice-interpolated* strategy is equivalent to the Joseph method. The Siddon method is simply called *siddon-line* and the *box-beam-integrated* strategy represents the method proposed by Shepp and Logan [Shep74] as depicted in Figure 3.5. Important for this work, is the *grid-interpolated* method, which represents the ray-casting approach in 3-D as *trilinear*. That's because that sample points are evaluated using a trilinear interpolation function. Looking at Figure b in [Xu06] the *Siddon-line* and *slice* strategies provide a very similar error profile. The *trilinear* strategy performs slightly better, but highlighting the *trilinear 2×* strategy that is superior to the previous mentioned strategies by applying a higher sampling rate. Xu *et al.* also compared interpolation methods with lower error rate by utilizing basis functions. These methods are not discussed as they require high computational efforts and therefore are less suitable for high performance implementations on GPUs.

Using the inverse projection matrix $F_p \in \mathbb{R}^{3 \times 3}$ the ray casting approach provides a flexible solution, see Equation 3.12. In addition, it provides a trade-off between accuracy and performance as this can be partially controlled via the sampling step-size. Therefore this algorithm is our first choice for the implementation on GPUs using CUDA.

Our ray casting approach using CUDA is based on the OpenGL counterpart detailed by Engel *et al.* in Chapter 7 of the book *Real-time Volume Graphics* [Enge06]. In the field of computer graphics ray casting is applied for the computation of

volume-rendering integrals and described (see book listing 7.1) in the following main steps:

```

Determine volume entry position
Compute ray direction
While (ray position in volume)
    Access data value at current position
    Compositing of color and opacity
    Advance position along ray
End While

```

In order to apply this approach as forward-projection operator in the field of medical image reconstruction, it has to be modified. Here the forward-projection needs to compute the line-integral of the object attenuation. Therefore, the computation of the color composition and opacity values of a graphics scene is needless. Instead only the evaluated attenuation values at the sampling positions are required to accumulate.

Algorithm 4.2: Forward-Projection for the p -th projection image.

```

input : volume  $\mu$ , matrix  $F_p$ , source position  $s$ , step-size  $\Delta l$ 
output: forward-projection  $\langle \mathbf{a}_r^{(p)}, \mu \rangle$ 

for  $\tilde{u}_1 = 1$  to  $U_1$  do
    for  $\tilde{u}_2 = 1$  to  $U_2$  do
        // calculate normalized direction vector from source  $s$  to
        // target  $t$  from  $\tilde{u}$ , see Equation 3.12
         $\hat{l} = \frac{F_p \cdot \tilde{u}}{\|F_p \cdot \tilde{u}\|_2}$ 

        // initialize attenuation value with zero
         $m = 0$ 
        // sample along the ray
        for each  $i$  representing the sample point  $x^{(i)} \in \overline{st}$  do
            // compute sample position
             $x^{(i)} = s + i \cdot \Delta l \cdot \hat{l}$ 
            // evaluate sample position and accumulate
             $m = m + \hat{\mu}(x^{(i)})$ 
        end
        // scale attenuation value according step size
        // and storing the result for the ray  $r$  resp.  $\tilde{u}$ 
         $\langle \mathbf{a}_r^{(p)}, \mu \rangle := m \cdot \Delta l$ 
    end
end

```

The applied ray casting algorithm is stated in Algorithm 4.2. Using a two dimensional example an intuitive description can be found in Figure 4.4. To determine the attenuation integral for a certain detector element, a straight line – respectively ray – needs to be calculated from the source position s of projection p towards the corresponding target position t of the measurement element r . The direction vector \hat{l} is calculated using the product of the inverse projection matrix F_p and the homogeneous detector position \check{u} , afterwards normalized to unit length. The shared source position s as well as the inverse projection matrix F_p for each projection p are extracted from the corresponding projection matrix B_p (see [Gali 03]). This facilitates a very flexible approach for arbitrary acquisitions geometries. This operation is followed by the equidistant evaluation of each sample point along the ray, which is accumulated to the initialized objective value. The specific amount of sample points is determined by the defined sampling step size Δl . Finally, the objective value has to be scaled with respect to the sampling step size Δl . Computing all measurement values of projection p results in the perspective forward-projection of the volume data.

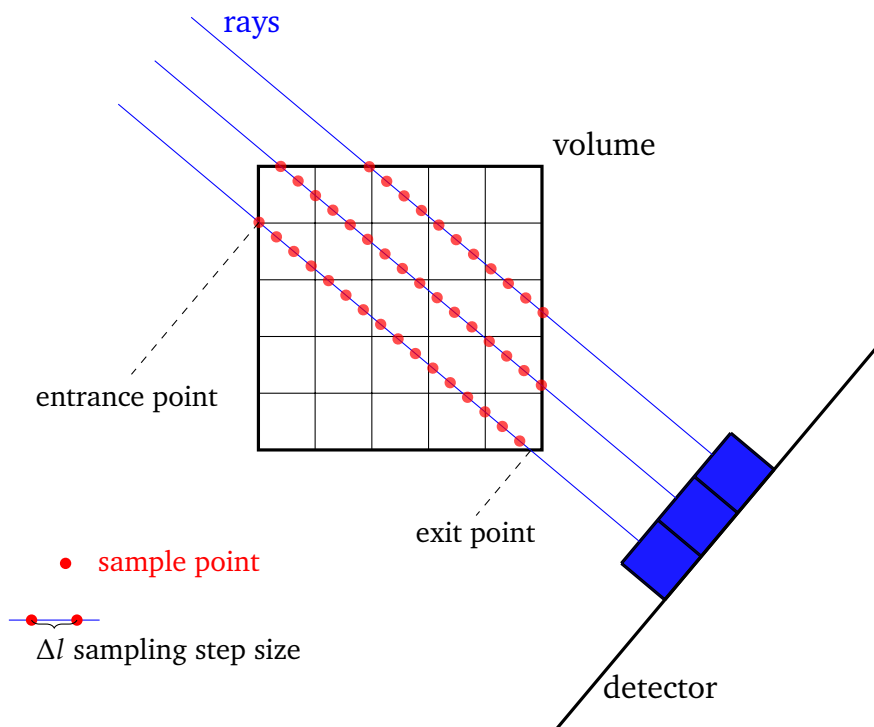


Figure 4.4: Ray casting principle visualized in 2-D for three rays. Each ray intersecting the volume is continuously sampled towards the detector element. The first sample point is defined by the position of the ray hitting the volume. Afterwards the next sample point is defined in distance by the sampling step size until the ray exits the volume.

Implementation

For the GPU implementation of the forward-projection operator using CUDA we suggested a straight forward parallelization scheme. An example for the implementation details and optimizations of the forward-projection is stated as CUDA kernel in the listings of Figure 4.5 and the utilized device function for the ray casting in Figure 4.6. In this example a 3-D texture including the 0.5 offset – required for correct textures access – is utilized to evaluate the sample positions.

Since each projection measurement is simulated with a single ray the CUDA grid- and block-layout is partitioned along the u_1, u_2 axes. Each CUDA kernel thread then approximates the line integral using an optimized version of the ray casting approach in Algorithm 4.2.

In the first step the normalized direction vector needs to be computed (see Fig. 4.5 line 14 – 16). This is done using the inverse projection matrix F_p and the detector position identified by block- and thread-index. Similar to the back-projection approach the inverse projection matrix F_p of size 3×3 elements, can be either stored in constant memory (see line 1) or loaded into shared memory at the very beginning. After the computation of the matrix-vector product, the resulting direction vector is normalized using the L^2 -norm (see line 18 – 21). Given the shared source position s and sampling step size Δl provided as kernel parameter or using constant memory, the ray casting device function can be executed (see line 23 – 24), which is explained in detail afterwards. The final unit normalization using L^2 -norm and the voxel size in each dimension – $\delta x_1, \delta x_2, \delta x_3$ – (see line 26 – 28) is required in order to adapt for unisotropic voxel sizes and world coordinate units.

Providing a flexible implementation the ray casting itself is defined in a device function `ray_cast()`, accessible by CUDA kernels only (see Fig. 4.6). In order to efficiently evaluate the sample points along the ray, only the computation of sample points inside the defined volume is required. Before the ray is equidistantly sampled inside the volume, the entrance and exit points for the ray hitting the volume are computed according to Siddon [Sidd85]. Instead of a concrete position the distance to the source position s along the ray is used and defined as $l \in \mathbb{R}_0^+$. Here the entrance position is defined using l_{min} as

$$\mathbf{x}_{entrance} = \mathbf{s} + l_{min} \cdot \hat{\mathbf{l}} \quad (4.3)$$

and the exit position using l_{max} by:

$$\mathbf{x}_{exit} = \mathbf{s} + l_{max} \cdot \hat{\mathbf{l}} \quad (4.4)$$

To compute the corresponding l_{min} and l_{max} for the given ray, the distance to the intersection of the ray with each side of the volume is calculated for non-zero elements of the direction vector $\hat{\mathbf{l}}$. Therefore the volume's minimal and maximal position in each dimension are required and stored in constant memory. This enables arbitrary volume positions including common ones, e.g., starting at the origin:

$$Vol_{min} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad Vol_{max} = \begin{pmatrix} X_1 - 1 \\ X_2 - 1 \\ X_3 - 1 \end{pmatrix} \quad (4.5)$$

Computing the maximal minimum-value and minimal maximum-value of the computed l values represents the shortest distance through the volume (see line 13 – 30). This equals the ray-volume intersection. Note that if an element of the direction vector \hat{l} equals to zero, then the vector is in parallel to the corresponding volume sides. In this case, no intersection points exist and therefore can be ignored. Given the computed l_{min} and l_{max} the ray casting itself is computed after initialization of the accumulation value. Inside the *Traversal Loop* (see line 33) the main component traverses along the ray and evaluates it at the calculated sample positions. As long as these sample positions lie between l_{min} and l_{max} the following four elemental computations are repeated, starting with $l = l_{min}$:

1. Compute the current sampling position according to: $\mathbf{x} = \mathbf{s} + (l_{min} + i \cdot \Delta l) \cdot \hat{l}$
2. Compute resp. read the attenuation coefficient for the current sampling position
3. Accumulate the evaluated value to the resulting value
4. Increment i resp. the current source distance with the given step size Δl

As the ray casting exits the *Traversal Loop* after the last sample point inside the volume was evaluated (see line 37), the resulting accumulated value needs to be scaled with the sampling step size Δl before returning the *ray_cast()*-function. Back in the forward-projection kernel, after scaling to world coordinate units the resulting value can be finally stored in global memory at the corresponding memory address.

The evaluation of the attenuation coefficient for each sampling position represents an import performance aspect due to the possible texture usage. Next, several different approaches for the evaluation are detailed and discussed before stating performance examples.

```

1  __constant__ float kpFp[3*3];    // projection matrix
2  __constant__ float kpSource[3];  // source position
3  __constant__ float delta_x[3];   // voxel size
4
5  __global__ void kernelForwardProject(
6      float* dpProj,
7      const int U1,
8      const float stepsize)
9  {
10     // compute projection coordinate
11     int u1 = __umul24(blockIdx.x, blockDim.x) + threadIdx.x;
12     int u2 = __umul24(blockIdx.y, blockDim.y) + threadIdx.y;
13     // compute ray direction
14     float l1 = kpFp[0] * u1 + kpFp[3] * u2 + kpFp[6];
15     float l2 = kpFp[1] * u1 + kpFp[4] * u2 + kpFp[7];
16     float l3 = kpFp[2] * u1 + kpFp[5] * u2 + kpFp[8];
17     // normalize ray direction
18     float norm = 1.0f / sqrtf( (l1*l1) + (l2*l2) + (l3*l3) );
19     l1 *= norm;
20     l2 *= norm;
21     l3 *= norm;
22     // compute forward projection using device function
23     float m = ray_cast(kpSource[0], kpSource[1], kpSource[2],
24                       l1, l2, l3, stepsize);
25     // normalize to world coordinate units using L2-norm
26     m *= sqrtf( (l1 * delta_x[0]) * (l1 * delta_x[0])
27               + (l2 * delta_x[1]) * (l2 * delta_x[1])
28               + (l3 * delta_x[2]) * (l3 * delta_x[2]) );
29     // compute storage position
30     unsigned int idx = __umul24(u2, U1) + u1;
31     // store result
32     dpProj[idx] = m;
33     return;
34 } // kernelForwardProject

```

Figure 4.5: Forward-projection kernel code for NVIDIA GPU using CUDA. The inverse projection matrix is accessed using constant memory for geometry computation. The ray casting is called separately using the defined device function. The problem is parallelized over u_1 - u_2 -plane.

```

1  __constant__ float kpVolMin[3]; // minimal volume position
2  __constant__ float kpVolMax[3]; // maximal volume position
3  texture<float, 3, cudaReadModeElementType> Vol;
4  // device function for ray casting
5  inline static __device__ float ray_cast(
6      float s1, float s2, float s3, // source position
7      float l1, float l2, float l3, // ray direction
8      float stepsize) // sampling step size
9  {
10     // compute l_min and l_max, entry and exit point
11     float l_min = MAXFLOAT; float l_max = 0;
12     // for each plane compute intersections
13     if (0.0f != l1) {
14         float reci = 1.0f / l1;
15         float temp0 = (kpVolMin[0] - s1) * reci;
16         float temp1 = (kpVolMax[0] - s1) * reci;
17         l_min = fminf(temp0, temp1);
18         l_max = fmaxf(temp0, temp1); }
19     if (0.0f != l2) {
20         float reci = 1.0f / l2;
21         float temp0 = (kpVolMin[1] - s2) * reci;
22         float temp1 = (kpVolMax[1] - s2) * reci;
23         l_min = fmaxf(l_min, fminf(temp0, temp1));
24         l_max = fminf(l_max, fmaxf(temp0, temp1)); }
25     if (0.0f != l3) {
26         float reci = 1.0f / l3;
27         float temp0 = (kpVolMin[2] - s3) * reci;
28         float temp1 = (kpVolMax[2] - s3) * reci;
29         l_min = fmaxf(l_min, fminf(temp0, temp1));
30         l_max = fminf(l_max, fmaxf(temp0, temp1)); }
31     // cast ray if it intersects the volume
32     float m = 0.0f;
33     while (l_min < l_max) {
34         float x1 = s1 + l_min * l1;
35         float x2 = s2 + l_min * l2;
36         float x3 = s3 + l_min * l3;
37         m += tex3D(Vol, x1+0.5f, x2+0.5f, x3+0.5f);
38         l_min += stepsize; }
39     // normalize to step size and return
40     m *= stepsize;
41     return m;
42 } // ray_cast

```

Figure 4.6: Ray casting device function code for NVIDIA GPU used in the forward-projection kernel. In this example the 3-D texture is used to represent the volume and evaluate the sample positions.

Sample point evaluation using different textures

One basic argument for the ray casting approach as a GPU implementation is the sample point evaluation and provided hardware acceleration using textures. Looking at the second step of the ray casting device function (see Fig. 4.6), the attenuation coefficients are evaluated for each sampling position. Therefore the CUDA API provides different memory models as discussed in Section 2.3.3. Due to the evolution of the CUDA API the presented approaches make use of the three different textures, introduced by NVIDIA one after another:

- **2-D texture array**
available with the first CUDA 0.8 API, starting in February 2007
- **3-D texture array**
provided by the CUDA 2.0 API, which was released in August 2008
- **2-D texture from pitch-linear memory**
in May 2009 introduced with the CUDA 2.2 API

2-D texture array

With the first release of the CUDA API – version 0.8 – NVIDIA did not provide any three dimensional textures. Compared to the OpenGL API at that time this was a clear drawback looking at the presented ray casting approach. To evaluate each sample point of the volume using a trilinear interpolation function (see Eq. 3.4) we applied the following workaround in the first realization.

Instead of a trilinear interpolation, we utilized two hardware accelerated bilinear interpolations using the 2-D texture array and performed the missing linear interpolation manually. Therefore the volume was distributed into a 2-D texture array slice-by-slice. This *texture atlas* [Enge 06] introduces additional computations as the offset for each slice inside the texture atlas has to be computed for each sample point. An illustration of the texture atlas is shown in Figure 4.7 as well as an exemplary device function for the workaround stated in the listing of Figure 4.8. Note that here three additional parameters are required. The number of volume elements in the x_1 - x_2 -plane represented by X_1 and X_2 . The third parameter is defined by the total number of volume slices contained in one column of the texture atlas. Given a maximum texture height of 2^{15} elements, as many slices as possible are distributed into one column of the texture atlas before extending the texture with another column. Here many possible distribution schemes exist, all given different advantages and disadvantages.

All schemes in common is a maximal volume size limitation of 2 GB for the 2-D texture array approach using floating point elements. If the volume exceeds 2 GB, one could use another texture instead of only one. However, this is not recommended as it introduces additional complexity. Furthermore, the partition of the volume into several parts for ray casting approach can introduce sampling artifacts respectively inaccuracies at the partitioning edges. Thus, we aimed for a one-texture solution. Looking at the initialization of the texture atlas each volume slice has to be copied into the 2-D texture array according to the selected scheme. In

the suggested approach each slice is copied separately with the appropriate position and offsets using the CUDA API function *cudaMemcpy2DToArray()*. As this does not look of importance, it has a significant contribution to the overall reconstruction performance, which we will detail later.

In summary, the 2-D texture array approach introduces additional complexity, additional computations, a complicated initialization, but enables almost arbitrary volume representation up to a size of 2 GB for 32-bit values.

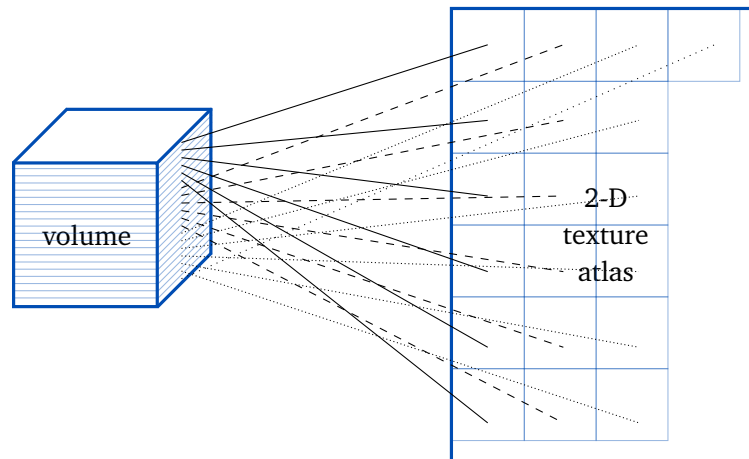


Figure 4.7: In order to represent a volume using a 2-D texture array each slice is spread onto a 2-D texture atlas. The exact order can be realized differently and therefore is exemplified.

3-D texture array

In August 2008 NVIDIA introduced their second major release of the CUDA API. With CUDA 2.0 NVIDIA also supported 3-D texture arrays besides OpenGL using their GPUs. For the ray casting approach this technical feature provided several benefits. On one side the sample point evaluation could be simplified to the CUDA API function call *tex3D()* providing a hardware accelerated trilinear interpolation as stated in the example in Figure 4.6. This eliminated the required additional computations. On the other side the initialization procedure could be trimmed from multiple CUDA API function calls for the volume slices to a single call using the CUDA API function *cudaMemcpy3D()* for the whole volume.

The 3-D texture array size for the utilized NVIDIA graphics cards is limited up to 2048 elements for each dimension. In total this allows a theoretical volume size of 32 GB for 32-bit float values which unfortunately cannot be provided by any graphics card yet.

```

1 // 3-D texfetch from texture atlas.
2 static __device__ float texfetch3D(
3     texture<float, 2, cudaReadModeElementType> Tex,
4     int StackHeight, int X1, int X2,
5     float x1, float x2, float x3)
6 {
7     int x1off, x2off;
8     float x_low, x_up;
9     // compute integer and fractional part of x3
10    int x3int = (int) x3;
11    float frac = x3 - x3int;
12    // compute and fetch value of lower slice
13    x1off = (x3int / StackHeight) * X1;
14    x2off = (x3int % StackHeight) * X2;
15    x_low = tex2D(Tex, x1 + (float)x1off, x2 + (float)x2off);
16    // compute and fetch value of upper slice
17    ++x3int;
18    x1off = (x3int / StackHeight) * X1;
19    x2off = (x3int % StackHeight) * X2;
20    x_up = tex2D(Tex, x1 + (float)x1off, x2 + (float)x2off);
21    // interpolate linear and return
22    return (x_low * (1.0f - frac)) + (x_up * frac);
23
24 } // texfetch3D

```

Figure 4.8: CUDA device function code for manual trilinear interpolation using a 2-D texture array. This is realized by two hardware accelerated bilinear interpolations combined with a linear interpolation in software. In this example the *StackHeight* represents the maximal number of volume slices inside a column of the texture atlas.

2-D texture from pitch-linear memory

CUDA 2-D and 3-D texture arrays have in common that they all are read-only (see Table 2.1) and cannot be accessed by a kernel in order to write data into. For the ray casting approach itself this is unproblematic, but is of importance to the proposed reconstruction approaches. Therefore another texture is utilized in the last approach. With the CUDA API 2.2 NVIDIA introduced 2-D textures from pitch-linear memory in May 2009. Compared to the swizzled memory structure of texture arrays, this type of texture provides a linear memory structure. Instead of the CUDA API function *cudaMallocArray()* the function *cudaMallocPitch()* is utilized to allocate the required memory.

In practice this memory does not differ from global memory. Thus it is also write-able with one limitation. As texture caches are not providing any coherency, this memory can be either used for writing or reading during a kernel execution. Despite the memory allocation the ray casting approach using a 2-D texture from pitch-linear memory does not differ from the proposed 2-D texture array approach.

Therefore the texture atlas and manual linear interpolation need to be utilized for the sample point evaluation. The initialization process is almost equivalent to the 2-D texture array approach. The only difference is that here the CUDA API function *cudaMemcpy2D()* is required to copy the data into the pitch-linear memory. The size limitations for this 2-D texture is equal to the one of the texture array, so the volume representation is limited for 32-bit values to a size of 2 GB. Given these three different approaches – using different textures – for the realization of the CUDA forward-projection operator based on ray casting, the performance measurement results are detailed next.

Performance

The presented performance results for the proposed ray casting approach are part of our published research [Wein 08]. As we could not identify significant performance differences between the approach using a 2-D texture array and the alternative using a 2-D texture from pitch-linear memory, we assume that both correlate strongly. Such both 2-D approaches are not distinguished in the following and referred as 2-D approach.

block-size	512 ² pixels	1024 ² pixels	2048 ² pixels
16 × 16	48.2	106	409
32 × 8	50.5	109	412
32 × 16	46.4	107	411
64 × 4	59.8	109	424
64 × 8	54.4	111	415
128 × 2	74.0	121	425
128 × 4	57.8	115	431
256 × 1	98.2	169	449
256 × 2	68.9	122	448
512 × 1	100	167	441

Table 4.4: Comparison of different block-sizes given by runtimes in seconds using the 3-D texture approach on the NVIDIA GeForce 8800 GTX. 400 projections and three different projection sizes at a step size of 0.25 of the voxel size and a total volume size of 512³ elements are compared.

Analogously to the back-projection approach, we start with setting the CUDA grid- and block-size for an optimal performance of this parallelization. As this setting has significant influence on the overall performance different settings need to be evaluated. A comparison of the ray casting performance for different block-sizes is stated in Table 4.4. Here, we utilized three different projection sizes for the comparison of the 3-D texture approach using CUDA 2.0. The performance results for ten different possible block-sizes are stated in seconds for 400 projections in total.

This reduces measurement inaccuracies and states an average value as different projection directions are used for each projection. The results for the 512^3 volume indicate that in most cases the best performance could be achieved using a block-size of 16×16 . We are going to use this block-size setting in this thesis when not stated differently.

In order to evaluate the drawback of the missing 3-D texture of early CUDA APIs we also implemented the ray casting approach using the OpenGL API. This was of main interest for performance comparison. Details about this implementation can be found in our paper [Wein08]. In Table 4.5 the performance results between the 2-D and 3-D approach using CUDA as well as the 3-D approach using OpenGL combined with a NVIDIA QuadroFX 5600 GPU are detailed. The runtime comparison in seconds are stated for three different projection sizes as well as four different amounts of projections. The runtimes clearly show the initial overhead required for a single forward projection, especially for the graphics API of OpenGL. At this sampling step size the 2-D approach using CUDA 1.1 takes at least twice the time compared to the 3-D approach using CUDA 2.0. The results suggest the 2-D approach is limited by shader computation and not by texture performance. More important the results prove that the performance for our 3-D texture ray casting approach using CUDA 2.0 is not only on par with the OpenGL API, it performs even slightly better.

# proj.	512 ² pixels			1024 ² pixels			2048 ² pixels		
	CUDA 1.1	CUDA 2.0	OpenGL	CUDA 1.1	CUDA 2.0	OpenGL	CUDA 1.1	CUDA 2.0	OpenGL
1	6.22	1.60	3.25	6.38	1.60	3.22	7.70	1.59	3.27
16	14.2	3.30	5.32	16.2	5.16	6.94	37.7	15.8	17.7
100	55.5	13.1	21.7	70.5	25.1	27.4	208	95.4	98
400	145	41.8	47.0	245	99.8	103	841	392	397

Table 4.5: Comparison of runtimes using the NVIDIA QuadroFX 5600 in seconds for a different number of projections as well as different projection sizes at a ray casting step size of 0.25 of the voxel size. CUDA 1.1 represents the 2-D texture approach. CUDA 2.0 as well as OpenGL utilize a 3-D texture. In all cases CUDA 2.0 performs best.

The G80 as well as the GT200 GPU architecture provide additional significant computation power in texture units as Schwarz showed in [Schw11]. Looking at the performance differences, this is clearly demonstrated by the comparison of the 2-D and 3-D approaches. As CUDA provides a scalable parallel programming approach [Lueb08] the comparison of the 3-D approach using the CUDA 2.0 and OpenGL API is illustrated in Figure 4.9. This highlights the linear scaling of the processing time depending on the projection count as well as the constant offset for the slightly slower OpenGL API. The different projection sizes can also be nicely differentiated by the performance slope as the computational complexity for twice the projection resolution quadruples.

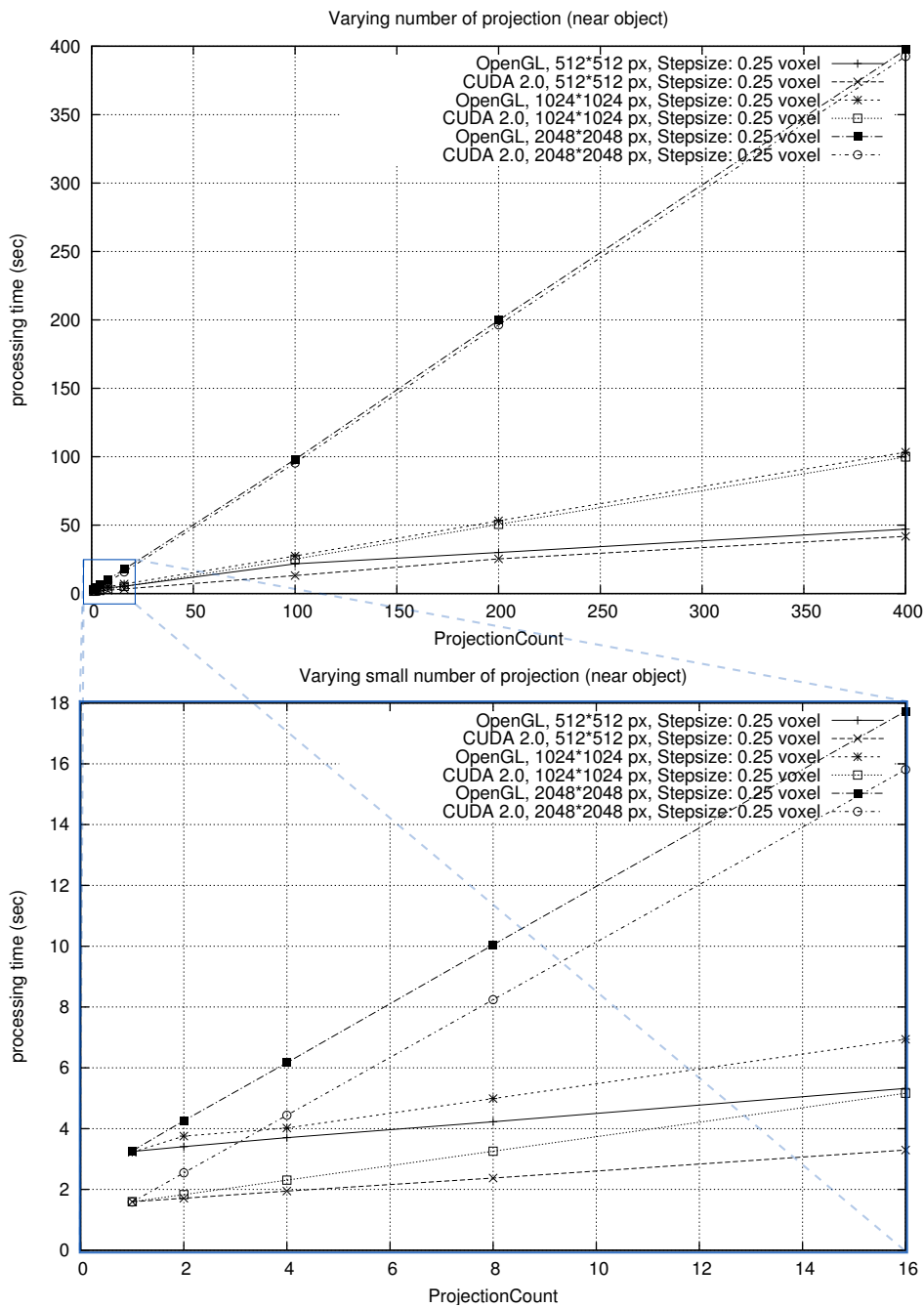


Figure 4.9: OpenGL and CUDA 2.0 API comparison with similar execution time behavior for varying projection count and size on a NVIDIA QuadroFX 5600. On the upper part the comparison is illustrated in total. The lower part shows the magnification for lower projection counts.

4.2 High Performance Medical Image Reconstruction Approaches

Given the introduced operators the realization scheme for the reconstruction algorithms is now focussed. With certain extensions and specifically added operations the introduced operators are combined to implement certain reconstruction algorithms detailed in Chapter 3. As this thesis focuses on high performance iterative reconstruction approaches, the implementation of the analytical reconstruction approach is repeated for completeness before introducing the realization scheme for the iterative approaches. Here the focus is on different realizations of the simultaneous algebraic reconstruction method and its variants, finishing with a well suited iterative statistical reconstruction approach.

4.2.1 Filtered Back-Projection and the FDK method

One of our earliest results in the field of high performance medical image reconstruction using the Compute Unified Device Architecture [Sche 07b] was the implementation of a practical cone-beam reconstruction algorithm by Feldkamp, Davis and Kress (Sect. 3.2.3), which is mostly used in angiographic imaging for 3-D reconstructions. From an implementation perspective the FDK method is very similar to the filtered back-projection reconstruction approach (Sect. 3.2.2). Therefore both implementations are handled together. The minor difference is due to a more advanced filtering step, applying a cosine weighting and a distance weight included in the back-projection.

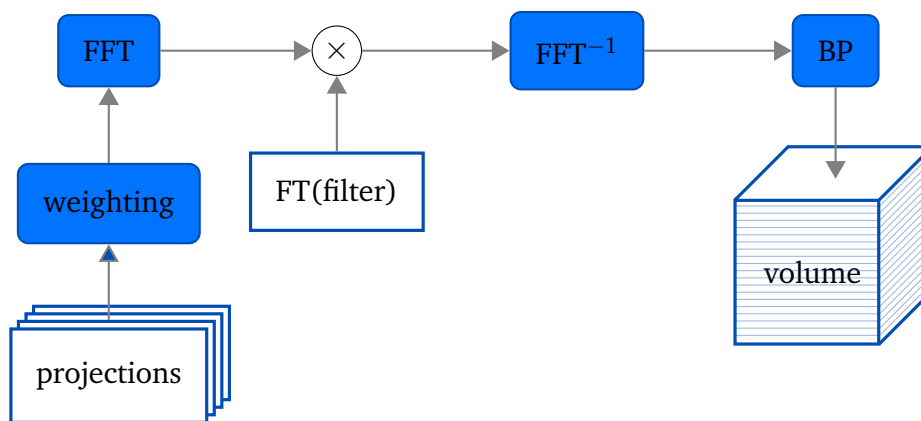


Figure 4.10: FDK block diagram example

Method

Looking at the realization scheme two operators are utilized for this reconstruction approach. For medical imaging the acquired measurement data is typically of sizes in which a filtering by convolution is of higher computational cost than performing the filtering step by multiplication in the frequency space. Therefore

the first operator used in this realization is the implementation of the Fast Fourier Transformation provided by NVIDIA in their accelerated libraries (Sect. 4.1.1) including the inverse FFT. The second operator is our optimized implementation of the voxel-based back-projection operator (Sect. 4.1.3).

The realization scheme is depicted in the block diagram shown in Figure 4.10. For the introduced approach the volume data remains on the GPU within global memory, while the projection data is transferred onto the GPU for each projection also into global memory. After transferring each projection onto the GPU, the appropriate weighting is applied and the cufft-library is used to transform each row of the projection data into frequency space. Here, the filtering can be applied by a simple multiplication of the Fourier transformed filter kernel. Afterwards the cufft-library is used again to invert the FFT and finally copying the filtered projection into the 2-D texture array using the CUDA function *cudaMemcpy2DToArray*. Given the filtered projection data as well as geometry information B_p the projection is back-projected on the objective volume – as described in the back-projection operator implementation. This process is then repeated for all projections and results in the aimed reconstruction approach detailed in Section 3.2.3 by Equation 3.18.

	time [s]	pps	fps
filtering			
NVIDIA GeForce 8800 GTX (CUDA)	3.00	138.00	
CELL processor 3.2 GHz (CBEA)	0.82	503.03	
back-projection			
NVIDIA GeForce 8800 GTX (CUDA, NN/LI)	7.06	58.64	72.52
CELL processor 3.2 GHz (CBEA, NN)	11.85	34.94	43.21
CELL processor 3.2 GHz (CBEA, LI)	20.99	19.73	24.40
data transfer (load projections / store volume)			
NVIDIA GeForce 8800 GTX (CUDA)	1.07 / 0.89		
CELL processor 3.2 GHz (CBEA)	0.00 / 0.00		
overall execution (filtering, back-projection and data transfer)			
NVIDIA GeForce 8800 GTX (CUDA, NN/LI)	12.02	34.44	42.60
CELL processor 3.2 GHz (CBEA, NN)	13.60	30.44	37.64
CELL processor 3.2 GHz (CBEA, LI)	24.04	17.22	21.30

Table 4.6: Performance results of filtering and back-projection in nearest neighbor interpolation (NN) and bi-linear (LI) interpolation mode.

Performance

Looking at the achieved performance results in Table 4.6 the original [Sche07b] comparison of the NVIDIA GeForce 8800 GTX GPU and CELL processor is stated. Before NVIDIA introduced the Compute Unified Device Architecture in 2007, the CELL processor was leading the fastest implementation on a commercial off-the-shelf hardware. Important to notice is that the main difference here was not only the performance, but also difference in development time. The development of

the first CUDA implementation was achieved in about three months whereas the development of the CELL processor implementation took more than one year.

Similar to the CPU results the CELL processors suffers from the missing texture units comparing the back-projection performance. Without the bilinear interpolation the performance difference shrinks to less than 68%. But the situation is different looking at the realized FFT performance. As the FFT does not require any interpolation the CELL processor could outperform the first CUDA capable devices by a factor of 3.65 in this application.

The advantages of the GPU can be really seen in the achieved performance results as well as the reduced development time. For further comparisons and details we refer to our results in [Sche 10, Sche 12] as well as [Hofm 09a, Hofm 10b]. After giving this motivating example for a GPU accelerated analytical reconstruction approach, the possible accelerations for SART and its variations are introduced.

4.2.2 Simultaneous Algebraic Reconstruction and its Variations

In 2009, we presented the first published performance results on CUDA-based 3-D iterative reconstruction using SART in the field of high performance medical image reconstruction [Keck 09a]. Later that year, extended results were published using new technical features for higher resolutions as well as lower memory requirements [Keck 09b].

Method

The key for high performance computations on the GPU – as described in Section 4.1 – lies inside the problem parallelization, such that computations are executed in a data-parallel fashion with optimal usage of the computational resources. Looking at the Algebraic Reconstruction Technique (see Sect. 3.3.2) such an optimal problem parallelization does not exist. This is due to the specific update rule that cannot be parallelized without a complicated computational sequence as well as introducing race conditions.

Instead the Simultaneous Algebraic Reconstruction Technique (see Sect. 3.3.3) seems well suited. Revisiting Equation 3.22 shows

$$\mu_j^{(k+1)} = \mu_j^{(k)} + \lambda \cdot \frac{\sum_{r=1}^R \left(\frac{d_\mu(p,r) - \sum_{c=1}^J a_{r,c}^{(p)} \cdot \mu_c^{(k)}}{\sum_{c=1}^J a_{r,c}^{(p)}} \right) \cdot a_{r,j}^{(p)}}{\sum_{r=1}^R a_{r,j}^{(p)}},$$

that the update rule can be divided into several parts. Mueller showed [Muel 98a] that the inner part of the SART update rule describes the **forward-projection** for each reading of projection p using Siddon's method [Sidd 85] (see Fig. 3.5). The difference to the measured attenuation $d_\mu(p,r)$ then states the **corrective term** that is finally back-projected onto the volume element μ_j . Looking at the presented (see Sect. 4.1.3) optimized voxel-based back-projection operator the corrective terms

for all voxels can be substituted as a corrective projection, such that each corrective projection needs to be back-projected onto the volume. For the computation of the corrective projection itself, the optimized ray-driven forward-projection operator (see Sect. 4.1.4) can be easily extended. Instead of storing the resulting value in global memory, this memory is initialized with the measured attenuation $d_\mu(p, r)$ such that in the final step of the forward-projection each thread computes the difference to this read value and multiplies the result with the relaxation factor λ before storing the result at the same corresponding memory address.

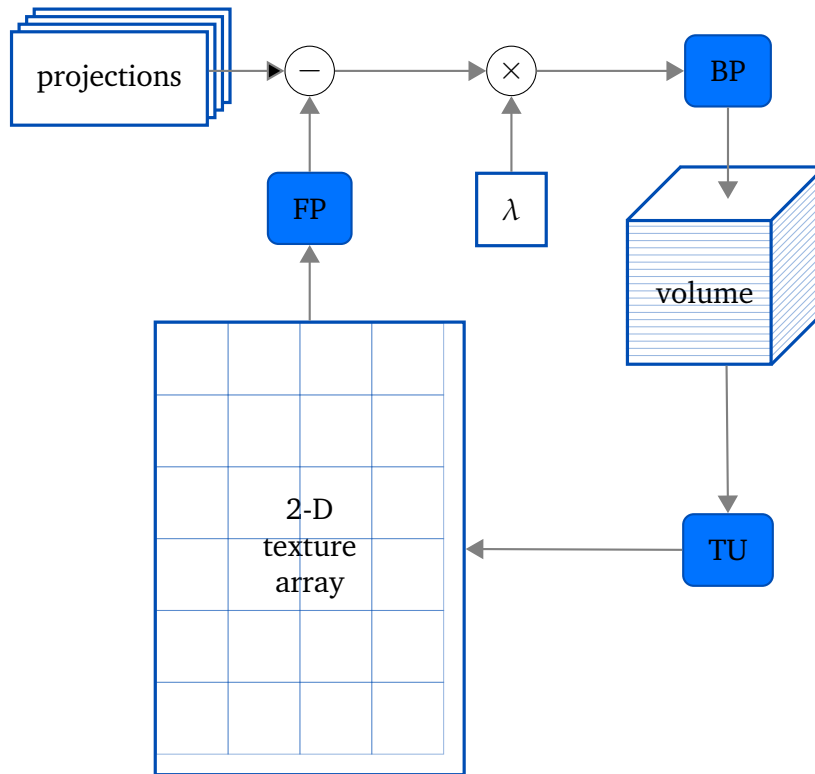


Figure 4.11: SART 2-D block diagram example

Given the voxel-based back-projection operator and our ray-driven forward-projection operator, the first realization scheme for CUDA 1.1 is depicted in the block diagram shown in Figure 4.11. This unmatched forward- and back-projector pair for iterative reconstruction was already investigated in research and hence we refer to the literature [Zeng 00]. Similar to the FDK method, the volume data used for the back-projection remains on the GPU within global memory. As the forward-projection operator utilizes textures, the volume data needs to be additionally kept in texture memory as well. For the CUDA 1.1 approach a 2-D texture array was utilized. In order to synchronize the texture to the global memory the Texture Update (TU) operator is used. For this approach copying slice by slice out of the volume remaining in global memory into the 2-D texture atlas (see Fig. 4.7) using the CUDA function `cudaMemcpy2DToArray` [NVID 07a]. As a drawback, this approach requires at least the doubled amount of memory for volume data on the GPU. Looking at the block diagram (Fig. 4.11) the SART is realized in the following steps.

For all projections, $\forall p \in P$:

1. Copy projection $d_\mu^{(p)}$ from host memory to global memory.
2. Compute corrective projection by performing FP with corresponding inverse projection matrix F_p , including final difference to $d_\mu^{(p)}$ and scaling using λ .
3. Copy corrective projection into 2-D texture array.
4. Perform BP of the corrective projection onto the volume using the corresponding projection matrix B_p .
5. Synchronize the 2-D texture atlas with the volume using the TU operator.

Repeating these steps for all projections P results in a single iteration of the aimed reconstruction approach detailed in Section 3.3.3 by Equation 3.22.

Looking at the variations of the SART the same approach can be used to realize the SIRT (Sect. 3.3.4) as well as the OS-SIRT (Sect. 3.3.5) by minor changes. For the SIRT realization the texture update in step 5 has to be skipped for all projections with respect to the last projection. During the computations the texture then represents $\mu_j^{(k)}$ while the volume contains the intermediate result of $\mu_j^{(k+1)}$ (see Eq. 3.24).

Analogously the OS-SIRT can be accomplished by modifying step 5 to be executed only at the last projection of each subset P_{OS} , such that the method implements Equation 3.25. Similar to the SIRT realization $\mu_j^{(k)}$ is represented by the texture and the intermediate result of $\mu_j^{(k+1)}$ contained in the volume.

Our first approach utilizes a 2-D texture array for the representation of the volume used in the forward-projection operator. As described in Section 4.1.4 we introduced three different approaches for the sample point evaluation of forward-projection operators using different textures. Therefore, two additional approaches for the SART and its variations are introduced.

Using the 3-D texture array instead of the 2-D texture array simplifies not only the sample point evaluation by utilizing hardware accelerated trilinear interpolation, but also the texture update operator could be improved noticeably. Instead of copying slice by slice, a single call of the CUDA function `cudaMemcpy3D` performs the synchronization of the 3-D texture array with the volume data. Combining the voxel-based back-projection operator and the forward-projection operator utilizing a 3-D texture array the second realization scheme for CUDA 2.0 is depicted in the block diagram shown in Figure 4.12.

Similar to the first realization scheme by modifying the texture update step the SART variations – namely SIRT and OS-SIRT – can be accomplished with the 3-D texture array approach.

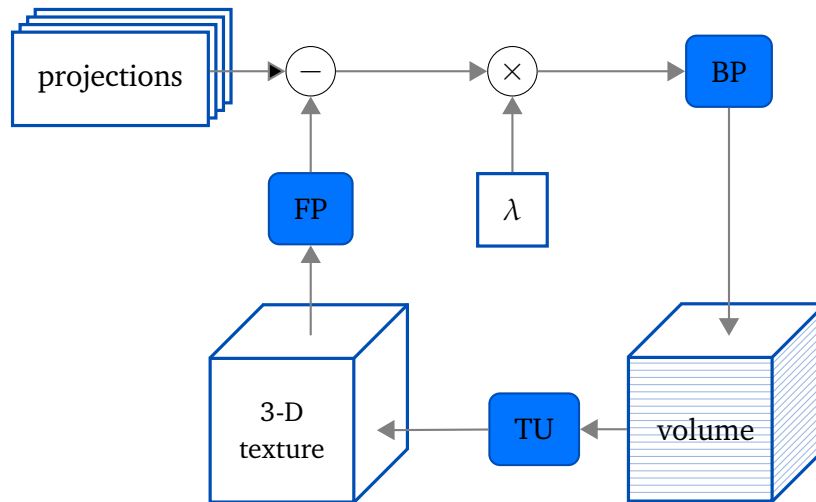


Figure 4.12: SART 3-D block diagram example

The last introduced methodology for the SART utilizes a 2-D texture from pitch-linear memory – introduced in May 2009 with the CUDA 2.2 API – inside the forward-projection operator. Analogously to the 2-D texture array approach the volume is represented inside a 2-D texture atlas (see Fig. 4.7) for the forward-projection. But in contrast to the CUDA 1.1 approach the back-projection operator is modified to utilize the same texture memory that can be write-able accessed in the global memory. Consequently, the computation of memory index to the first voxel of the x_2 -column has to be adapted for the texture-atlas layout. The few additional parameters and computations for this adaption are incorporated similar to the *3-D texfetch from texture atlas* as stated in the listing shown in Figure 4.8. Due to the fact that the back-projection operator as well as the forward-projection operator are executed with separated kernel calls, GPUs texture caches are invalidated in-between and therefore the non-provided cache coherency of the texture is unproblematic. The introduced texture update operator in this case is non-existent and the resulting third realization scheme of the SART using CUDA 2.2 and a 2-D texture from pitch-linear memory is illustrated in the block diagram of Figure 4.13. Using a 2-D texture from pitch-linear memory allows higher volume resolutions, e.g., exceeding a slice resolution of 2048^2 . Additionally, the memory requirements – compared to the previous approaches – can be reduced.

Looking at the possible implementations of SIRT and OS-SIRT our last approach has pros and cons, as the texture update operator is missing. Compared to the above presented, this approach demands only half of the volume memory requirements. On the other side, a simple modification by the texture update operator to implement the SART variations is not easily possible using a 2-D texture from pitch-linear memory. Hence these variations are not treated here.

Performance

Given these three different approaches for SART and its variations the performance results are elaborated. The first performance results cover the CPU-GPU compar-

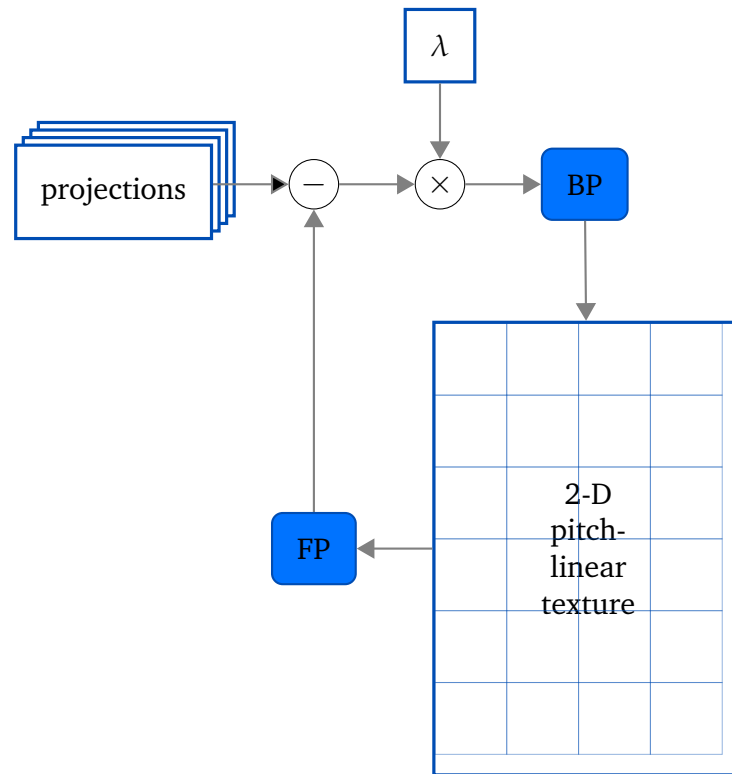


Figure 4.13: SART reconstruction block diagram using a 2-D texture from pitch-linear memory

ison as published in [Keck09a]. For the comparison an existing multi-core based *Reconstruction Framework* by Holger Kunze [Kunz07] was utilized for CPU performance measurements. On the GPU side the reconstruction is performed on a NVIDIA QuadroFX 5600 graphics card providing 1.5 GB memory (see Table 2.2). The phantom dataset consists of simulated projections, generated with the DRASIM simulation tool [Stie00]. A short-scan of a C-arm-CT system is represented by 228 projections each size of 256×128 pixels and the iterative reconstruction yields a $512 \times 512 \times 350$ volume. In order to achieve a sub-voxel sampling in the forward-projection step we used a step-size of 0.3 of the isotropic voxel-size as described in the test dataset description in Appendix A.3.

One of the major results of our contribution besides the GPU acceleration is that a significant amount of time for the first two approaches – using texture arrays – is spent on the texture update operator. As this operator is required in order to use the hardware-accelerated interpolation provided by textures, the time spent on the texture update operator can be significantly reduced by switching to the OS-SIRT method. Table 4.7 shows the achieved performance for 20 iterations of the CPU-based SART reconstruction as well as our optimized GPU implementations using CUDA 1.1 and CUDA 2.0. Further improvements are observed for the reconstruction times for the OS-SIRT method using different subset sizes and 20 iterations alike. As the CPU implementation cannot utilize textures, they do not need additional memory for the forward-projection operator nor require a texture update operator.

Therefore, reconstruction times for the SART and all OS-SIRT measurements are identical.

hardware/ method	Intel Core2Duo DualCore 2 GHz	2×Intel Xeon QuadCore 2.33 GHz	QuadroFX 5600 CUDA 1.1	QuadroFX 5600 CUDA 2.0
SART	32968	6630	4234	844
OS-SIRT(2proj.)	”	”	2435	661
OS-SIRT(5proj.)	”	”	1359	551
OS-SIRT(7proj.)	”	”	1156	530
OS-SIRT(10proj.)	”	”	998	514

Table 4.7: Comparison of iterative reconstruction times in seconds for SART and OS-SIRT (for 20 iterations each). Two different CPU systems as well as one GPU with two different CUDA versions.

For the CUDA 1.1 approach, the texture update operator represents a non-negligible share of the computational time. We measured 476 seconds to synchronize a 512^3 volume 414 times with the 2-D texture atlas (see Fig. 4.7). This results in approximately 1.15 seconds for a single texture update for this volume size. Using a 3-D texture array instead in our CUDA 2.0 approach, this can be improved by a factor of 10 as a texture update can be performed in approximately 0.11 seconds.

In comparison to the CUDA 2.0 API our SART implementation using CUDA 1.1 wastes approximately 1 second per texture update for a typical volume size of 512^3 floating point volume elements. If the number of forward- and back-projections between two texture updates is increased slightly, the reconstruction speed is improved while the convergence rate remains almost at the same level, but won't be as fast as for the SART (see Sect. 3.1.2). **The novelty that a reconstruction algorithm with a slower convergence can actually beat another algorithm with a faster convergence in terms of reconstruction performance is one of the important results of our research.** The trade-off between convergence and speedup was also examined by Xu *et al.* They come to the same result, that the algorithm with a slower convergence can actually be computed much faster. We refer to the literature [Xu 10] for a detailed analysis of the convergency behaviour for SART and OS-SIRT in CT.

In Table 4.7 we compared the reconstruction times on three different systems. First, an off-the-shelf PC equipped with an Intel Core2Duo E6600 processor running at 2 GHz, second, a workstation with two Intel Xeon E5410 QuadCore processors at 2.33 GHz and a NVIDIA QuadroFX 5600 running CUDA 1.1 and CUDA 2.0 are utilized. The SART implementation using CUDA 1.1 is the slowest implementation on the GPU. Yet it is more than 7.5 times faster than the PC and 50% faster than the workstation. Employing the ordered subsets optimization yields another speedup of over 4. SART using a 3-D texture array for interpolation (CUDA 2.0) is even a bit faster. Using the OS-SIRT approaches again results in a total speedup of up to 64 times and 12 times faster compared to the PC (2 cores) and the workstation (8 cores) respectively.

The second performance results cover the extension to *high resolution iterative CT reconstruction* using the GPU as published [Keck09b]. The fastest iterative implementations on graphics cards use 3-D texture arrays to exploit hardware-accelerated trilinear interpolation. However, the size of textures arrays is subject to technical limitations (see Table 2.1). As particular applications of computed tomography require high slice resolutions, e.g., in 3-D mammography (see Sect. 1.1.2), 3-D texture arrays are inapplicable for such applications. Alternatively a 2-D texture array can be used instead of the 3-D texture array as in the first realization scheme, but the texture update operator causes a significant loss of performance.

In our third realization scheme we utilize a new feature – 2-D pitch-linear texture – of the released CUDA 2.2 framework to improve the Simultaneous Algebraic Reconstruction Technique on GPUs. In order to compare the performance of the last scheme we extend the experiment from our first results [Keck09a]. The achieved performance results are stated in Table 4.8 in extension with the benefits and technical limitations of the three presented realization schemes. Additionally the performance results for NVIDIA’s second generation of CUDA-capable devices (see Table 2.2) are stated for the NVIDIA Tesla C1060 graphics card.

volume	512 ³ voxels			
hardware	QuadroFX 5600		Tesla C1060	
volume	2-D	3-D	2-D pitch-	2-D pitch-
repres. (FP)	texture array	texture array	linear texture	linear texture
volume	global mem.	global mem.	global mem.	global mem.
repres. (BP)	linear	linear	texture atlas	texture atlas
device mem. [MB]	700	700	350	350
TU	yes	yes	no	no
CUDA version	≥CUDA 1.1	≥CUDA 2.0	≥CUDA 2.2	≥CUDA 2.2
SART	4234	844	1488	955

Table 4.8: Comparison of iterative reconstruction times in seconds (for 20 iterations each).

Compared to the 3-D texture array approach, the third realization scheme is about 76% slower. Using NVIDIA’s 2nd Tesla generation, the Tesla C1060, the reconstruction time for the last approach can be further reduced from 1488 to 955 seconds. Overall, the third realization scheme states a new balance between performance, limitations in resolution and memory requirements. Specific applications with higher resolutions as well as performance results for this approach will be detailed in Chapter 5.

Given these performance results for SART and its variations, we conclude the section on algebraic reconstruction techniques. The optimized reconstruction performance for our realization scheme was outlined. The advantage of using the texture memory of current graphics cards to perform the most time-consuming parts

of an iterative reconstruction technique effectively on the GPU using CUDA 1.1 and CUDA 2.0 was compared. Apparently, in CUDA 1.1 the time consuming texture update operator dominates the overall reconstruction time. This can be dramatically relieved using 3-D texture arrays as introduced in the CUDA 2.0 API. Therefore, the impact on a Ordered Subset approach using a 3-D texture array is lower, compared to the 2-D texture array approach. For higher resolutions, the advantage of using 2-D texture lookups from pitch-linear memory to overcome the 3-D texture size limitation for volume representation are exemplary stated.

4.2.3 Maximum Likelihood Reconstruction

Another contribution in the field of high performance medical image reconstruction covers the third group of reconstruction algorithms – namely statistical reconstruction methods – detailed in Section 3.4. As described our work is based on the paper [Lang 95] by Lange *et al.* Looking at the introduced approaches of statistical reconstruction, we focus on the high performance realization of the maximum likelihood convex algorithm as detailed in Section 3.4.3 and suggested by Lange *et al.* in the discussion, see Section IV.

Method

For a high performance realization scheme, it is essential to optimally dissect this algorithm, such that the computations are executed in a data-parallel fashion with optimal usage of the computational resources. Revisiting the update rule of the ML-Convex algorithm in Equation 3.43:

$$\mu_j^{(k+1)} = \mu_j^{(k)} + \frac{\mu_j^{(k)} \sum_{p=1}^P \sum_{r=1}^R a_{r,j}^{(p)} \left[I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle} - d_I(p, r) \right]}{\sum_{p=1}^P \sum_{r=1}^R a_{r,j}^{(p)} \langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle}},$$

shows the required computations to be realized. Before detailing our efficient realization, numerical details of this equation are treated. In our experiments we determined that this equation introduces numerical instabilities for floating-point implementations, which resulted in image artifacts. Furthermore, the multiplicative version of the ML-Convex in Equation 3.44 also indicated numerical instabilities. **Therefore, an important result of our research on high performance statistical reconstruction is a mathematical transformation of Equation 3.43 that provides higher and sufficient numerical stability for the GPU implementation using floating-point arithmetics.** The numerical problem in the update rule is caused by the product of the intensity I_0 and the exponential term $e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle}$. As the intensity value is typically of a high value and the exponential term can result in a very small value for high attenuation integrals. The multiplication then is numerical not stable. In the suggested solution, the intensity value is raised into the power

term as $\log I_0$. This provides higher numerical stability as the logarithmic intensity is much smaller and incorporated as a sum in the exponent. The transformed update rule as implemented then states:

$$\mu_j^{(k+1)} = \mu_j^{(k)} + \frac{\mu_j^{(k)} \sum_p \sum_r a_{r,j}^{(p)} \left[e^{\log I_0 - \langle a_r^{(p)}, \mu^{(k)} \rangle} - d_I(p, r) \right]}{\sum_p \sum_r a_{r,j}^{(p)} \langle a_r^{(p)}, \mu^{(k)} \rangle e^{\log I_0 - \langle a_r^{(p)}, \mu^{(k)} \rangle}} \quad (4.6)$$

This update rule is different to the update rule of the SART, but also shows important similarities. Here, the **forward-projection** as described in Section 4.2.2 is used in the numerator as well as twice in the denominator of Equation 4.6. In the numerator the difference to the measured intensity is computed as a **corrective term** as well as a **normalization value** in the denominator. Both values are then back-projected, but not directly as in the realization scheme of our SART approach.

Analogously to our SART implementation, we depict the problem in a projection-wise fashion. Looking at all readings R of projection p , the corrective term of the numerator can be interpreted as a corrective projection named as numerator projection. Similar the normalization values for projection p in the denominator are declared as denominator projection. As we distinguish between the upper and lower part of the fraction in Equation 4.6 a numerator and a denominator volume is utilized besides the volume residing in the texture array. For the realization scheme, we use the introduced operators for the forward-projection (see Sect. 4.1.4) and the back-projection (see Sect. 4.1.3) and modify both according to the following. For a projection p , the parallel kernel of the forward-projection is extended to compute each corrective term of the numerator projection similar to the SART. In addition the computed **forward-projection** is used to compute and store the normalization value in parallel in the denominator projection. The required $\log I_0$ is appended into the kernel call. Both resulting projections are copied afterwards into a 2-D texture array each.

While both projections could be independently back-projected onto the respective volume, the performance can be improved by modifying our back-projection operator such that the numerator and denominator projection are back-projected using a single kernel onto the appropriate volume. Therefore, the geometrical computations are minimal and a higher workload for the back-projection kernel can be achieved.

Similar to SIRT the ML-Convex is a non-successive method, meaning that the update for each pixels is computed using all projections. Therefore, the alternating computations of the FP and BP operator are repeated for all projections P . After the last projection the actual update value including the fraction is computed by a new update operator (UP). The update operator is parallelized identically to the back-projection operator. For each voxel the update operator reads the original value $\mu_j^{(k)}$ from the texture array, as well as the appropriate value from the numerator and denominator volume out of global memory. Using these input values the update rule

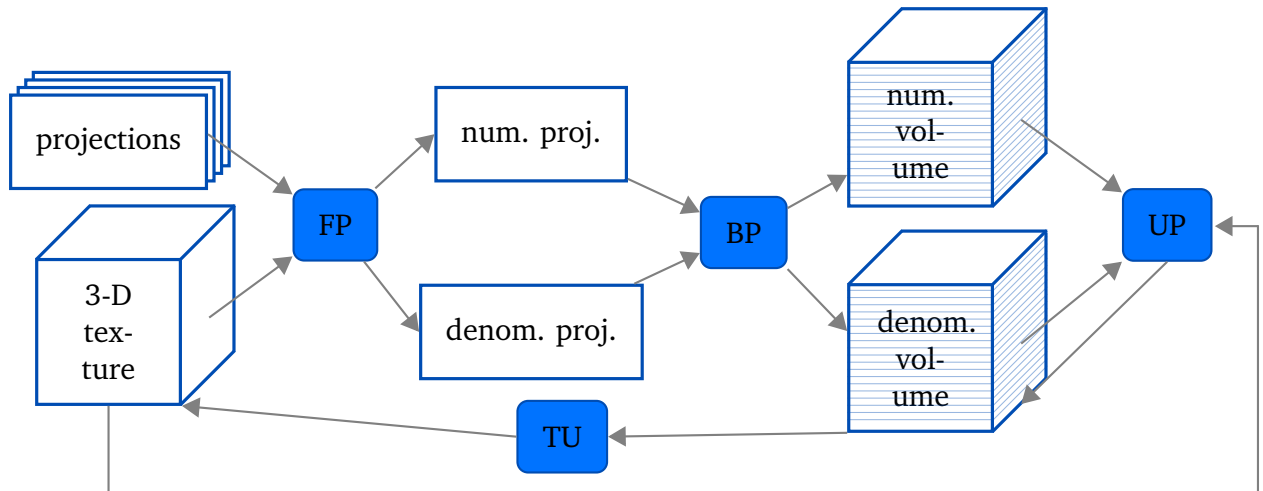


Figure 4.14: ML-Convex 3-D block diagram example.

from Equation 4.6 then is computed for each voxel and $\mu_j^{(k+1)}$ stored in a target volume, e.g., using the denominator volume. Following this, the texture update (TU) operator synchronizes the target volume to the utilized texture, completing one iteration of the ML-Convex algorithm. As a drawback, this approach requires at least three times the volume data on the GPU. Statistical reconstructions require certain conditions like the non-negativity constraint. These constraints as well as a relaxation factor λ can be easily incorporated into the update operator.

The first realization scheme using a 3-D texture arrays is depicted in the block diagram shown in Figure 4.14. One iteration of the ML-Convex reconstruction is realized in the following steps.

For all projections, $\forall p \in P$:

1. Copy intensity projection $d_i^{(p)}$ to global memory of the numerator projection from host memory.
2. Compute numerator projection and denominator projection by performing the forward-projection with corresponding inverse projection matrix F_p . This includes the computation of the corrective term and normalization value (see Eq. 4.6).
3. Copy numerator and denominator projection into the corresponding 2-D texture array.
4. Perform BP of both projections onto the appropriate volume using the corresponding projection matrix B_p .
5. For the last projection P , perform the UP operator that computes $\mu_j^{(k+1)}$ using $\mu_j^{(k)}$ from the texture memory as well as the corresponding value from the

numerator and denominator volume. Store $\mu_j^{(k+1)}$ in the target volume in global memory.

6. For the last projection P , synchronize the texture with the target volume using the TU operator.

Due to the fact that the 3-D texture array may limit the resolution, we also present a second approach for higher resolutions similar to the SART approach. Here, it is of minor difference if a 2-D texture array or a 2-D texture from pitch-linear is used. On the one hand, the memory requirements cannot be reduced as it is necessary to provide three times the volume on the GPU for our realization scheme. The back-projection cannot utilize the pitch-linear memory as the original memory $\mu_j^{(k)}$ resides in the texture memory that is required for the update operator. On the other hand, the texture update operator is used only once for all projections and therefore the performance difference is marginal. The second realization scheme for higher resolutions using a 2-D texture atlas is depicted in the block diagram shown in Figure 4.15.

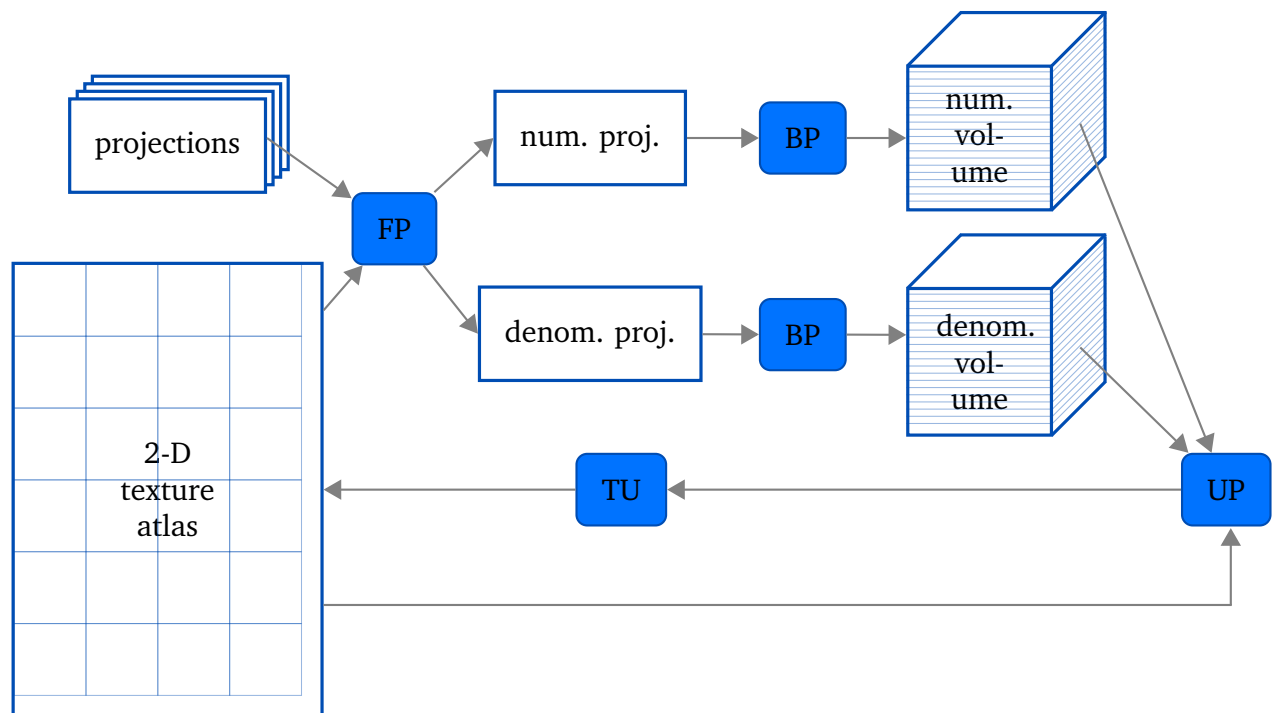


Figure 4.15: ML-Convex 2-D block diagram example

Performance

Given these two realization schemes for the maximum likelihood convex algorithm, a first look at the performance results is stated. In Section 3.5 we discussed that

SART and ML-Convex have the same complexity, but differ in the amount of required volume updates. The performance benefit of the GPU versus CPU was already stated in the last section. Hence, we compare our SART implementation with the ML-Convex approach for a simulated data head phantom in this performance evaluation. The dataset consists of 90 projections over 180° . Each projection has a size 1024^2 readings. The target volume was set to a size of 512^3 floating-point elements. Details for this test dataset are stated in Appendix A.4. Repeating the SART experiment, the reconstruction times in total of 20 iterations were measured to compare the average performance of the ML-Convex algorithm.

volume	512 ³ voxels	
hardware	Tesla C1060	
volume repres. (FP)	3-D texture array	3-D texture array
req. device mem. [MB]	1024	1536
approach	SART	ML-Convex
reconstruction time	600	452

Table 4.9: Comparison of iterative reconstruction times for the SART and ML-Convex approach in seconds (for 20 iterations each).

As discussed in Section 3.5 the ML-Convex has a slower convergence than the SART and therefore in practice requires more iterations for the reconstruction. As the computational performance is compared in Table 4.9, both measurements state the same number of iterations. Looking at the achieved performance for both approaches using a NVIDIA Tesla C1060 GPU, the ML-Convex realization scheme is almost 25% faster than our SART approach using a 3-D texture array. This is mostly due to the highly reduced number of texture updates. On the other side, this indicates that the additional computations for both corrective projections as well as the twofold back-projections are implemented very efficiently as the update operator itself represents only a small part of the required computations.

This concludes the high performance implementation of the maximum likelihood convex algorithm as well as the section on high performance medical image reconstruction approaches.

4.3 OpenCL Comparison

As part of our research the Open Computing Language (see Sect. 2.4.1) was also investigated. This was not the main focus of this research and therefore our experience in OpenCL in comparison to CUDA is summarized. Details on the comparison and analysis can be found in our research results, presented at the annual SPIE Medical Imaging Conference [Sieg 11b] and the German workshop BVM – Bildverarbeitung für die Medizin – on medical image processing [Sieg 11a].

The results additionally detail the comparison for the proposed back-projection and forward-projection operators for a Multi-core CPU system as well as a AMD/ATI GPU. Here, we want to focus on the difference between the OpenCL implementation, portability as well as performance differences for NVIDIA GPUs.

Looking at the performance results for the NVIDIA Tesla C1060 stated in Figure 4.16, both operators can be accelerated with the GPU compared to a CPU implementation. In detail, a general portable implementation of the back-projection operator is possible using OpenCL with the drawback of a highly reduced performance. Here the underlying hardware is not fully taken into account. Similar to our CUDA implementation the OpenCL implementation can be improved if the parallelization scheme is adapted, but these optimization techniques (see Sect. 4.1.3) are introduced together with the lack of portability. The OpenCL implementation with such optimizations is 2.7 times faster than without adaptations, but 10% slower compared to the analog CUDA implementation of the back-projection operator. For the forward-projection operator this difference is only about 4 percent.

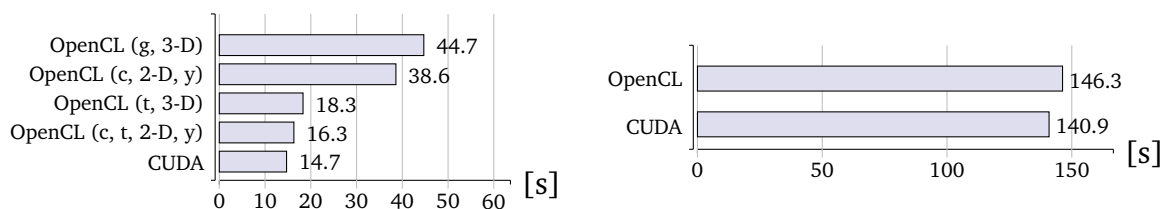


Figure 4.16: OpenCL vs. CUDA performance comparison in seconds on a NVIDIA Tesla C1060. Left: Back-projection runtimes using four different OpenCL and an optimized CUDA implementation. For the OpenCL implementation (g) means generic, (c) coalesced, (t) textures units, (v) vectorized. (x) and (y) denote the index of the innermost loop. Right: Forward-projection operator using a 3-D texture comparing OpenCL and CUDA.

This indicates that OpenCL in general can be used as a language for acceleration using GPUs. In order to achieve high performance this comes to a lack of a main claim of OpenCL namely portability¹. Without the portability – between different GPU manufacturers and CPUs – CUDA instead can be used on NVIDIA GPUs providing a slightly higher performance. As NVIDIA stepped back for future OpenCL development and support of their GPUs, clearly CUDA is the only viable solution for high performance implementations using NVIDIA graphics cards.

¹<http://en.wikipedia.org/wiki/OpenCL>

This concludes the chapter on high performance medical image reconstruction that depicts the main research contribution of this thesis. Next, the value of our contribution is conveyed in form of two clinical applications with focus on performance and image quality considerations.

4.4 Conclusion

In practice, medical image reconstruction algorithms often have to be executed with very strict time constraints. GPU technology proved to provide sufficient computational power for the implementation of analytical reconstruction methods and can meet those time constraints. For the class of iterative algorithms the computational demand is a multiple of analytical ones and therefore it is questionable whether those more complex algorithms can be practically applied. GPU acceleration can be utilized for medical pre- and post-processing algorithms. For the realization of GPU-accelerated iterative image reconstruction, the introduced operators for back-projection and forward-projection are key. In order to achieve high performance these operators are implemented taking different aspects into account. Each algorithm is divided into lightweight and independent tasks which can be executed in parallel. Different technical features of the GPU hardware are utilized. The implementation pays attention to the underlying hardware and GPU architecture such that a suitable parallelization scheme for high performance is achieved. Given those operators the key for the actual realization of the iterative reconstruction methods is the presented suitable decomposition of the algorithms. Consequently, the operators are properly combined to implement the reconstruction algorithm. Therefore, the forward- and back-projection operators are modified and additional operators introduced.

In terms of performance, the SART implementation and its variants highly depend on the available technical features of the GPU such as 3-D texture arrays or 2-D textures from pitch-linear memory. Using these technical features a significant speedup can be achieved on a single GPU compared to a desktop CPU or dual CPU workstation.

An implementation of the statistical reconstruction algorithm which achieves high performance on the GPU is most challenging. In addition to the selection of an appropriate decomposition and parallelization also numerical issues have to be considered. Hence, the presented approach is based on the decomposition of a reformulated update rule and a tricky combination into sequences of parallel implementations. This is key for the achieved high performance results.

Finally, we can state that the usage of OpenCL as an alternative to CUDA in general is possible. OpenCL also allows execution on CPUs as well as GPUs from AMD/ATI. However, performance comparable to the CUDA implementations is only achievable if the underlying hardware is taken into account. This leads to a lack of portability, one of the major claims of OpenCL. The achieved acceleration using CUDA as well as the presented performance results demonstrate the possible application of iterative reconstruction methods using the high computational power of off-the-shelf hardware.

Clinical Applications

Given the technical and theoretical background as well as a detailed description of the different scientific contributions, clinical applications are the main focus of this chapter. In order to emphasize the impact and relevance of this thesis, we exemplarily show two clinical applications including performance comparison and image quality aspects.

In focus of the first clinical application is the interventional C-arm imaging as described in Section 1.1.2. Afterwards 3-D mammography is covered as the second clinical application, where breasts are imaged using a limited angular scan range (see Fig. 1.6).

5.1 Interventional C-arm CT

In interventional C-arm CT digital flat-panel detectors are used to acquire digital X-ray projections for image guided treatment as well as rotational projections in order to compute cross-sectional images supporting the interventional procedure. Therefore, manufacturers provide different imaging systems, all given different capabilities. With the modern C-arm CT system, the Siemens Artis ZeeGo, an industrial joint-arm robot is used to move a pair of X-ray emitter and a detector mounted on a C-like structure with a high degree of freedom. This allows traditional image acquisition trajectories like a circular arc, but also enables non-standard trajectories (see [Denn 08]).

The datasets utilized in Section 4.2 were based on a geometry typical for C-arm CT. As shown in Section 4.2.1 high performance implementations using the GPU enabled reconstruction times of a few seconds for analytical reconstruction methods with application in C-arm CT. In this field the CELL processor already enabled on-the-fly reconstruction for different analytical reconstructions [Kach 07, Sche 07c, Sche 07a] before. As sufficient performance could be achieved within this clinical application, more advanced reconstruction methods may be facilitated if clinical benefits can be provided.

Looking at the simultaneous algebraic reconstruction technique, this is still in focus of research and was investigated also by H. Kunze (see Section 7.2 of [Kunz 07]). In order to demonstrate this reconstruction technique, different examples for simulated data motivate possible applications.

The first example presented is based on the high resolution approach for iterative CT detailed in Section 4.2.2. As published in [Keck 09b], we utilized simulated

phantom projections of the Catphan CTP528 phantom in order to assess the image quality of our approach. Therefore, 400 projections of 1024×128 pixels each were generated with DRASIM [Stie 00], given a pixel size of 0.3×0.7 millimeters. Details for the datasets can be found in Appendix A.4. The iterative reconstruction of the CTP528 phantom is illustrated in Figure 5.1 showing all 21 line-pairs.

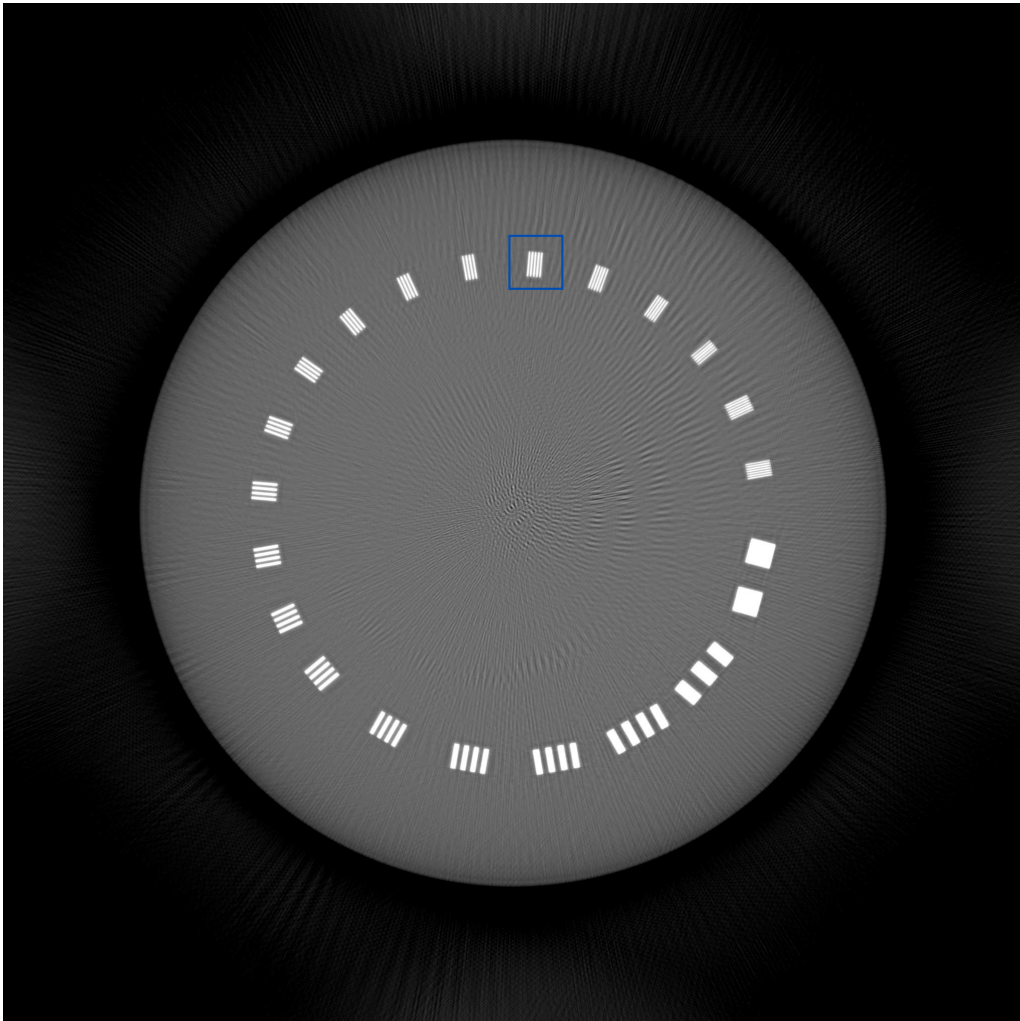


Figure 5.1: Iterative reconstruction of the Catphan CTP528 phantom using simulated projections and 20 iterations of the SART using the GPU. The 16th line pair is marked with the rectangle. Greyscale window $[-1000 \text{ HU}; 1000 \text{ HU}]$.

In order to compare the image quality as well as reconstruction performance, the phantom was reconstructed using different resolutions and reconstruction setups as stated in Table 5.1 all with SART. The table also includes the achieved performance as measured for 20 iterations each. Due to the 2 GB memory limitation of a 2-D texture from pitch-linear memory, the number of slices for the highest resolution of 3072×2048 pixels was reduced to 50.

hardware	Tesla C1060			
volume	512×512	1024×1024	2048×2048	3072×2048
size	$\times 100$	$\times 100$	$\times 100$	$\times 50$
voxel size	0.4×0.4	0.2×0.2	0.1×0.1	0.075×0.1
in mm	$\times 0.1$	$\times 0.1$	$\times 0.1$	$\times 0.1$
device memory required [MB]	100	400	1600	1200
SART rec. time	1166	2407	11353	4951

Table 5.1: Comparison of iterative reconstruction times in seconds for SART for high resolutions (20 iterations each).

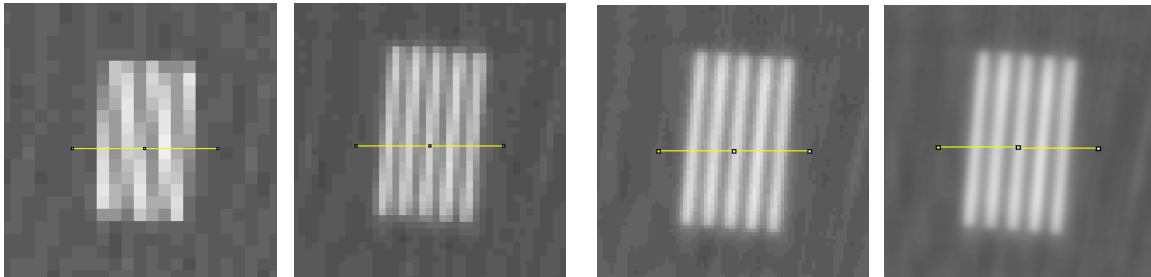


Figure 5.2: Iterative reconstructions details of the 16th line pair of the Catphan CTP528 for different slice resolutions from left to right: 512^2 pixels, 1024^2 pixels, 2048^2 pixels and the highest resolution of 3072×2048 pixels. The visualized yellow line across the elements illustrates the measurement line used for the line profiles.

The increased reconstruction resolution suggests an increase in image quality. For comparison, the reconstruction results of the 16th line pair of the phantom are illustrated for each setup in Figure 5.2 together with the measured lines (marked yellow). These attenuation coefficient on these lines are additionally visualized for comparison as line profiles in Figure 5.3.

The shown reconstruction examples and corresponding line profiles clearly indicate the benefit for higher resolutions. As stated in Table 5.1 the presented enhanced GPU accelerated SART reconstruction for high resolution volumes can be performed in a similar performance range as our first SART approach using a 3-D texture array. In contrast the memory requirement for the high resolution approach is lower. Also the reconstruction can provide an improved image quality if higher resolutions can be utilized. However, a total reconstruction time of up to 190 minutes still points out the limitation for clinical use. For certain research applications this techniques can be utilized with extensive reconstruction times, but clearly not for clinical routine, yet.

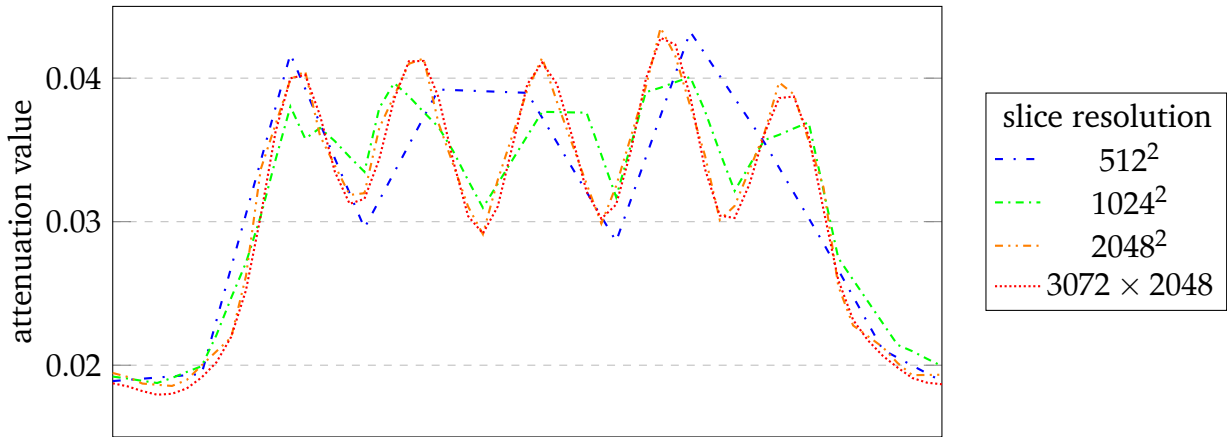


Figure 5.3: Different line profiles of the line across the 16th line pair of the Catphan CTP528 phantom are compared. The profiles correspond to different reconstruction resolutions using SART.

hardware	Tesla C1060				
	algorithm	FDK	SART	ML-Convex	
number of iterations	n.a.	20	20	50	200
reconstruction time	3	566	461	1142	4549

Table 5.2: Performance comparison for different reconstruction methods using the Tesla C1060 GPU. The dataset consists of only 90 projections over a limited angle of 180°. The performance results are given in seconds.

Another example to motivate iterative reconstruction methods, is given by the following experiment. Using a head phantom and the DRASIM simulation tool a C-arm CT acquisition over a limited angle of 180° for the low number of 90 projections was simulated (see Sect. A.5). Using this data the reconstruction results are compared for a single slice of a 512³ volume in Figure 5.4. Therefore, the reference phantom as well as a standard FDK reconstruction are shown together with different iterative results. Looking at the centered top of the head phantom the FDK reconstruction clearly indicates the missing data of the circular arc as these acquisitions are typically using an trajectory over 220° (depending on the fan-angle). The SART can provide only a minor improvement in this area. Looking at the ML-Convex a clear improvement is visible compared to the other methods and therefore provides a better reconstruction in this area. Given the performance results as stated in Table 5.2, the drawback of these methods is clear. The reconstruction time reaches from a few seconds up to 75 minutes for 200 iterations of the statistical reconstruction method. The statistical reconstruction provides the slowest convergence and therefore requires most iterations.

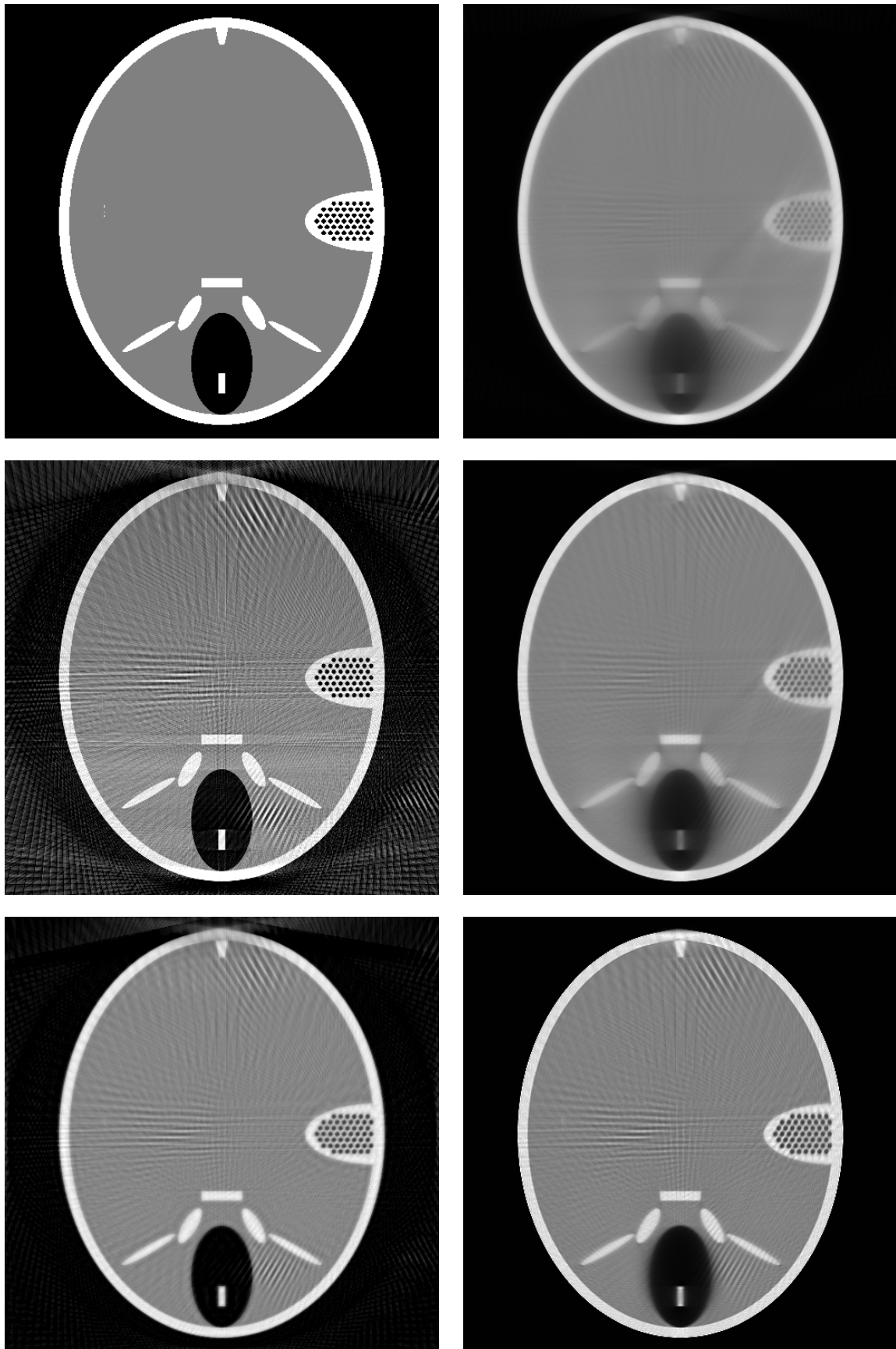


Figure 5.4: Slice 258 of the reconstructed head phantom. Left images from top to bottom: Phantom reference slice, FDK reconstruction and SART reconstruction with 20 iterations. Right side from top to bottom is the ML-Convex reconstruction with 20, 50 and 200 iterations. Greyscale window $[-1000 \text{ HU}; 1000 \text{ HU}]$.

5.2 3-D Mammography

The second clinical application is motivated by advanced screening methods for breast cancer. In 3-D mammography the digital breast tomosynthesis technique aims to provide depth information by the acquisition of several projections over a limited angle for the reconstruction of focussed slices respectively a 3-D volume. However, it suffers from incomplete data and poor quantum statistics limited by the total dose absorbed by the breast. Breast tissue has a relatively high sensitivity to radiation and therefore the total dose needs to be as low as possible. Our research in this field was in close collaboration with clinical experts. A clinical discussion of the results is out of scope of this thesis and the reader is referred to the publications by Jerebko *et al.* [Jere 10a, Jere 10b] for further details, analysis of the achieved image quality as well as the advantages of iterative reconstruction techniques in this field.

Jerebko *et al.* showed the image quality advantages [Jere 10b] of the ML-Convex algorithm for 3-D Mammography. The ML-Convex algorithm [Lang 95] is identical to our presented approach in Section 4.2.3. Thus, we motivate a possible clinical application by the achieved performance results and comparison. Furthermore, a few image examples of the presented realizations are stated to illustrate the underlying modality as well as give an insight into the challenges and possible solutions.

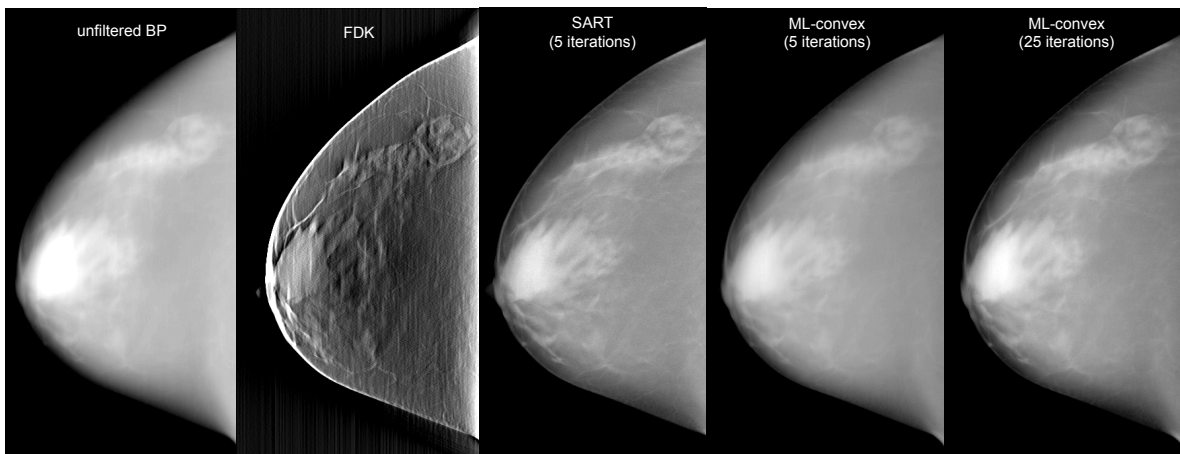


Figure 5.5: 3-D mammography reconstructions for different algorithms. From left to right. The unfiltered back-projection is stated as a possible initialization for iterative reconstruction methods. The FDK reconstruction is elevated due to the different impression caused by the filtering step and the specific filter. This is followed by two iterative reconstructions, the SART and ML-Convex reconstruction results after 5 iterations. On the right side the reconstruction result after 25 iterations of the ML-Convex is illustrated. Greyscale window $[-900 \text{ HU}; 300 \text{ HU}]$.

All presented reconstruction approaches of Section 4.2 can be applied in the field of 3-D mammography. Figure 5.5 shows the different impressions of the breast tissue structures achieved by analytical, algebraic and statistical reconstruction algorithms. The FDK approach uses a Mammography specific filter kernel to provide better sensitivity in terms of contrast resolution, but loses information about the true tissue density. The unfiltered back-projection can be used for an improved initialization for iterative reconstruction methods. In order to impart the difference to a constant initialization, the unfiltered back-projection is also illustrated.

hardware	2×Intel Xeon QuadCore 2.33 GHz	QuadroFX 5600			Tesla C1060
resolution	high	normal	normal	high	high
texture	n.a.	3-D	2-D lin.	2-D lin.	2-D lin.
rec. time	2:12 h:m	102s	232s	290s	198s

Table 5.3: SART reconstruction performance comparison between CPU and GPUs for two different resolutions. The times stated are measured for 5 iterations. Due to size limitations different texture approaches are used in the GPU results.

With the benefits of GPU acceleration analytical reconstructions in 3-D mammography can be achieved within a few seconds. Therefore, we look into the computational more advanced SART for performance comparison in the field of breast tomosynthesis. We described in Section 1.1.3 that 25 projections à 2816×3584 pixels are acquired by the Mammomat system. For the reconstructions two typical resolutions are used. The normal resolution uses a voxel size of $0.17\text{mm} \times 0.17\text{mm} \times 1\text{mm}$, resulting in a total volume of $829 \times 1435 \times 60$ voxels. The high resolution setting applies a voxel size of $0.085\text{mm} \times 0.085\text{mm} \times 1\text{mm}$ such that an identical size requires $1658 \times 2870 \times 60$ voxels (Sect. A.5). The achieved reconstruction performance is compared in Table 5.3 for 5 iterations of the SART.

In order to underline the proposed high resolution approaches, we look into the following 3-D mammography example. Figure 5.6 shows two SART reconstructions both with the same amount of iterations. On the left side the typically resolution for the clinical environment is used. On the right side the full detector resolution is utilized for the reconstruction as introduced beforehand.

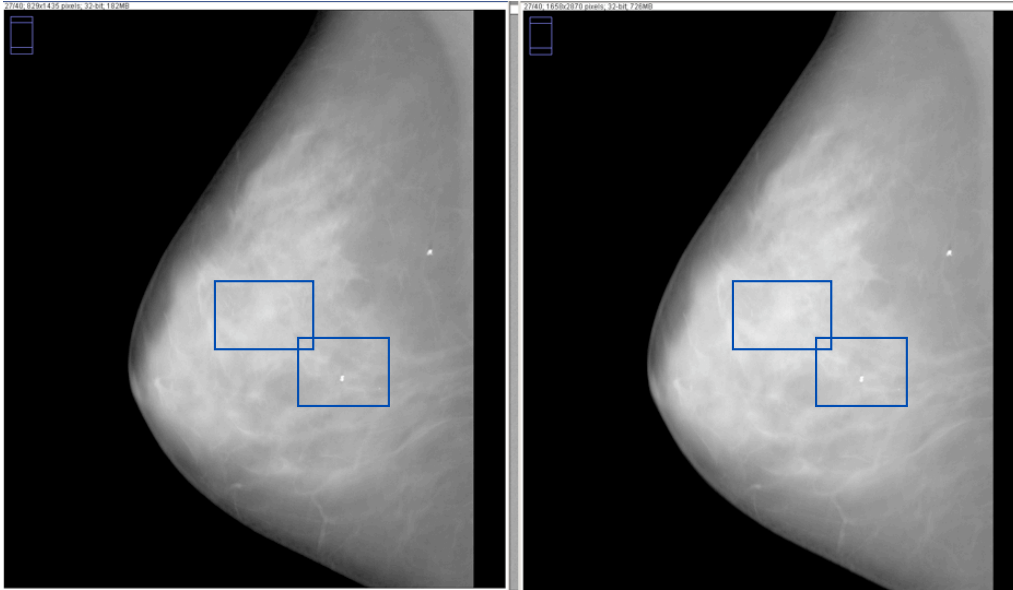


Figure 5.6: SART comparison for regular and high resolution. On the left side a slice with the typical clinical resolution is shown. On the right side the high resolution was used for the reconstruction. In both images the rectangles mark the same magnification areas. Greyscale window $[-650 \text{ HU}; 50 \text{ HU}]$.

For non-experts both images seem to be identical. Therefore, two magnifications are used in combination with a different window in order to illustrate the important differences. First, looking at the bright nodule in the lower right of the image center, the magnification in Figure 5.7 clearly states an improved illustration of the shape of this calcification.

Even more interesting is the second example. Looking at the magnification in Figure 5.8 of the center image region using a different window, some micro-calcification can be indicated for the typical resolution. Looking at the high resolution reconstruction the two right-handed small micro-calcification turn out to be a cluster of many tiny micro-calcifications. This is important as a cluster of micro-calcifications can be an important indicator for an expanding breast cancer.

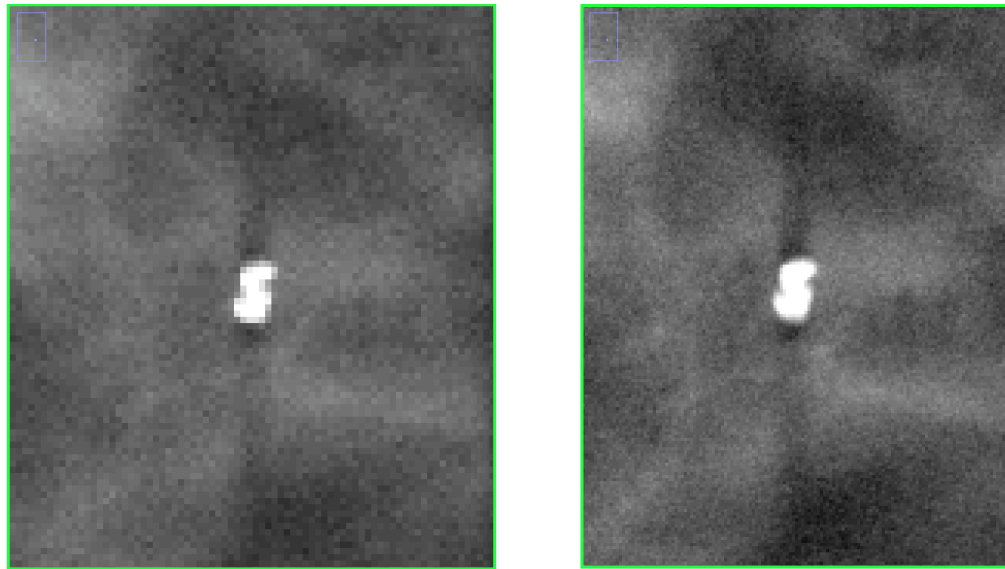


Figure 5.7: Magnification of the SART resolution comparison for the bright calcification. On the left the typical resolution was used. On the right side the high resolution SART reconstruction was applied. Greyscale window $[-650 \text{ HU}; 50 \text{ HU}]$.

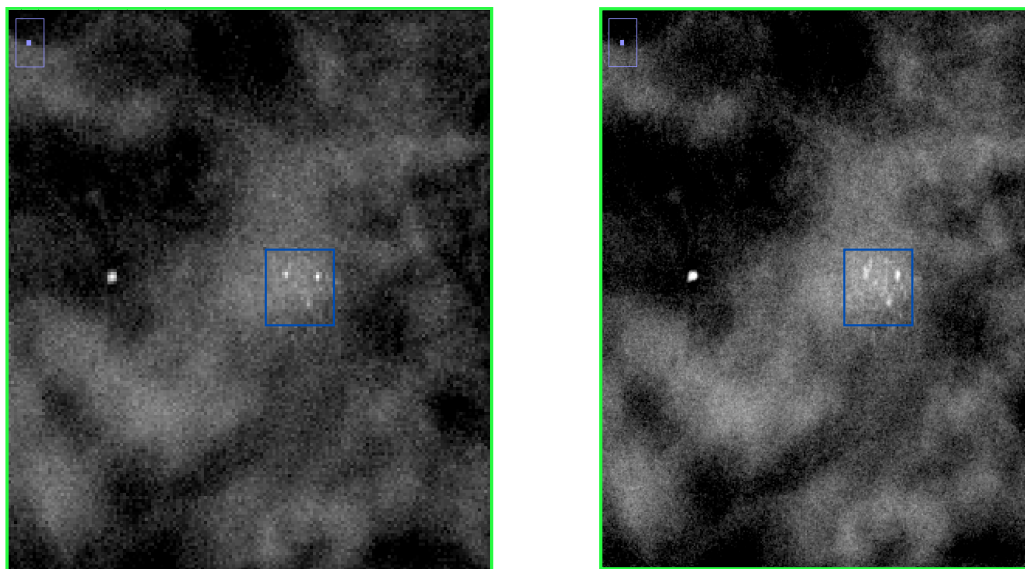


Figure 5.8: Magnification of the SART resolution comparison for micro-calcification using a different visualization window. The high resolution on the right side clearly indicates the benefits compared to the typical resolution on the left. Greyscale window $[-200 \text{ HU}; 200 \text{ HU}]$.

Given these benefits of the SART and higher resolutions the second iterative algorithm is investigated. Jerebko *et al.* presented a detailed analysis on the image quality of the ML-Convex algorithm in [Jere 10b]. The proposed ML-Convex approach requires more iterations due to a slower convergence (see Sect. 3.5). However, the performance per iteration compared to the SART approach is slightly faster due to the reduced amount of texture updates. In Table 5.4 the performance results are stated for a clinical 3-D mammography with a reconstructed voxel size of $0.085\text{mm} \times 0.085\text{mm} \times 1\text{mm}$ and thereby defined resolution of $1144 \times 2048 \times 38$ voxels (see Sect. A.6). Note that this enables the 3-D texture approach.

hardware	Tesla C1060		
	algorithm	SART	ML-Convex
number of iterations	5	5	25
reconstruction time	36	29	138

Table 5.4: 3-D mammography SART vs. ML-Convex performance comparison. Reconstruction performance in seconds.

The convergence of the ML-Convex depends on various factors and therefore the required amount of iterations cannot be defined. In our experience with a good initialization 5 iterations often are sufficient. However, even 25 iterations with a reconstruction time of 2.5 minutes are still applicable for clinical routine. The proposed approach can be easily extended in the forward projection operator to compute the log-likelihood defined in Equation 3.29. This enables a comparison of the convergence behaviour with a fixed dataset and different initializations. In Figure 5.9 four different initializations are compared over 50 iterations of the ML-Convex using a plot of the log-likelihood. Three different constant initializations are compared with an initialization using an unfiltered back-projection. The plot and magnification indicate the benefits of the unfiltered back-projection initialization as well as the convergence for each iteration. For a higher number of iterations the advantages become minor.

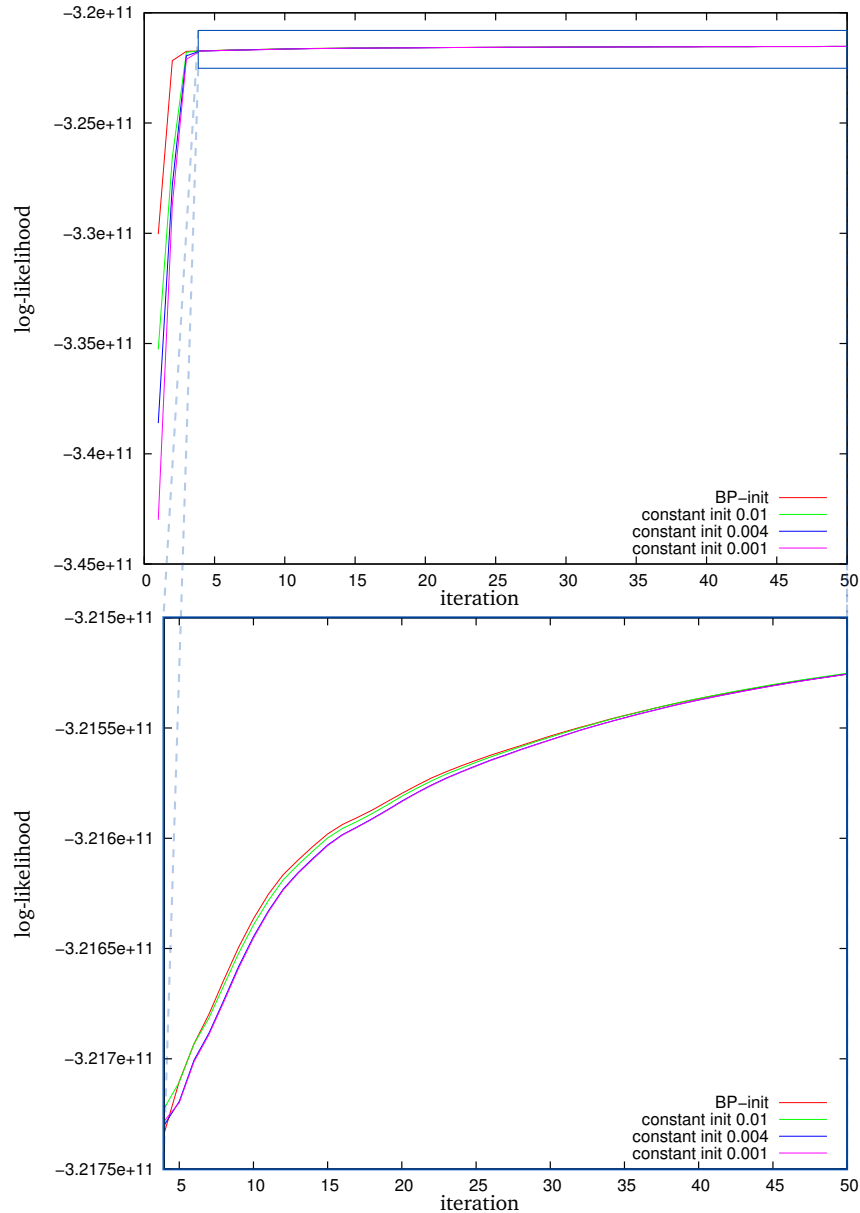


Figure 5.9: Log-likelihood for the ML-Convex reconstruction over 50 iterations is plotted on the top. The different initializations influence the appropriate starting points of the log-likelihood. An unfiltered back-projection (BP-init) initialization is more similar to the aimed reconstruction result and therefore shows a much higher log-likelihood than the constant initializations. On the bottom the comparison is plotted starting with 5 iterations. This details the log-likelihood slope as well as details the decreasing differences between the different initializations. The log-likelihood is computed according to Equation 3.29.

5.3 Conclusion

In this chapter we have shown two clinical applications of iterative X-Ray CT, both requiring high performance reconstruction. With interventional C-arm CT and 3-D mammography two clinical applications are exemplified for the performance and image quality aspects of the presented research. For interventional C-arm CT the ML-Convex algorithm can improve image quality in case of missing data. However, the actual computational cost in this case is still prohibitively high for implementation in clinical routine, yet it enables further clinical research on this method. In case of algebraic reconstruction the higher resolution approaches are verified with the reconstruction of a simulated Catphan phantom. The performance comparison of our approaches underlines the high performance of GPUs as well as motivates the use of this class of reconstruction methods for interventional C-arm CT.

In the second clinical application, 3-D Mammography, the image quality aspects have already been investigated by Jerebko *et al.*. Our presented approaches can provide a significant performance improvement and therefore enable iterative reconstruction for this time sensitive screening method. The higher resolutions are justified with the illustration of reconstructed micro-calcifications. This exemplifies the possible application of iterative reconstruction methods in practice. In the last chapter this work is summarized and an outlook on future medical image processing on GPUs is given.

Summary and Outlook

In order to provide an outlook for GPU accelerated medical image processing the essence of the first five chapters is summarized. This includes the main contributions as well as conclusions of this work. The thesis is then finished with a more general outlook for GPU acceleration and recent developments within the field of high performance computing related to medical image processing.

6.1 Summary

The relevance of this work is introduced in correlation to the history of CT and its rising usage and complexity. The exponential increase of measured signals per seconds over the last four decades (see Sect. 1.7) illustrates the first substantial motivation. Additionally, this is reaffirmed by the manufacturers' development and introduction of specialized acceleration cards in the field of medical image processing over the last decades (see Sect. 1.2.1).

With the introduction of unified shaders and the CUDA programming language in 2007, GPGPU quickly has drawn interest from researchers and developers. The immense compute power and enormous memory bandwidth of GPUs together with a parallel programming approach enabled the utilization of high computational power from off-the-shelf hardware. Since then GPU accelerated medical image processing strongly increased its share in many fields of application including medical image reconstruction.

The second substantial motivation is founded on the algorithmic complexity of reconstruction algorithms (see Sect. 3.5). The trend towards iterative reconstruction algorithms expresses an additional demand of higher computational power in order to meet performance requirements of numerous medical imaging systems.

In the interest of presenting our scientific contribution, the basic knowledge of our research is detailed in the second chapter on GPU Programming and in the third chapter on Medical Image Reconstruction. In the first section of Chapter 4 two different examples of GPU acceleration using CUDA are demonstrated. This is followed by the two important basic operators for iterative reconstruction methods. Hence the forward- and back-projection are detailed. The proposed implementations, variants, optimization techniques and resulting performance examples demonstrate the achieved high performance. In example our presented back-projection implementation running on a NVIDIA QuadroFX 5600 GPU is 29 times

faster than a highly optimized CPU implementation running on an Intel Core2Duo Conroe CPU (2 Cores at 2.4 GHz). These GPU accelerated algorithms using CUDA proof their advantages against other processors – like the CPU and the CELL processor – as well as to other GPU APIs, like OpenGL. This points out the suitability of GPUs in combination with CUDA for medical image reconstruction algorithms. Afterwards the presented high performance reconstructions are detailed in combination of the proposed operators together with modifications and additionally required operators. As a representative of analytical reconstruction methods, the FDK approach was realized combining the GPU accelerated FFT together with the presented back-projection operator. The resulting approach could improve the performance compared to the CELL broadband engine by a factor of 2 (see Sect. 4.2.1). While this seems modest, the difference has to be seen in the development time and efforts spent for such performance results as well as the fact that the CELL processor was one of the most efficient and powerful processors at that time.

Looking at iterative algorithms, the Simultaneous Algebraic Reconstruction Technique and its variations were realized combining the two main operators – with minor modifications of the forward-projection – together with an additional update operator. Three realizations detail differences in performance as well as technical limitations (see Sect. 4.2.2) using the GPU. Again our approaches demonstrate the achieved high performance for these iterative algorithms using different technical features and circumventing texture size limitations. Comparing a single GPU with a multi-core CPU system states a significant performance improvement of a factor of 12 by the introduction of GPU acceleration within this field of – iterative – algebraic reconstruction.

Considering the more advanced reconstruction group of statistical reconstruction algorithms, we were able to present the ML-Convex reconstruction approach also utilizing the accelerated operators. Through use of specific partitioning of the required computations together with minor modifications the algorithm was successfully realized on the GPU. In order to achieve this, the mathematical background is of importance. Numerical instabilities could be circumvented by mathematical conversion. While the theoretical computational complexity is equivalent to SART, the achieved performance clarifies the high performance statement. The direct comparison to SART shows that the algorithm can slightly perform better than the accelerated SART due to reduced execution of the additional update operator as well as an introduced higher computation load for the modified parallel kernels and further optimizations. In total the ML-Convex provides a slower convergence and therefore requires more iterations, reducing the benefit in terms of total reconstruction time.

Besides the main scientific contributions, the clinical application of the presented approaches is exemplified in the fifth chapter. For two different modalities the performance results are stated together with image quality aspects. These examples indicate the possible usage of GPU accelerated reconstruction approaches for medical imaging products. Especially for 3-D mammography the presented high resolution approach for the SART demonstrates clinical usability as well as diagnostically benefits in case of micro-calcifications.

In summary we demonstrated the immense performance benefits of GPUs in the field of iterative medical image reconstruction. Minor technical limitations were indicated and technical developments were incorporated to further improve performance or circumvent limitations. In order to realize medical image reconstruction algorithms using CUDA, we indicated the importance of a suitable partitioning and parallelization of these algorithms as well as highly optimized implementations in our presented research.

6.2 Outlook

Giving an outlook for the future of medical image processing on GPUs is a quite difficult task, but also important to lead further research directions in this field. Many different economical aspects influence future development of GPUs as well as CPUs. However, we can identify a clear trend for these technologies.

Looking at the evolution of CPUs the central processing unit was designed to process dozens of different tasks simultaneously with a low latency. The graphics processing unit instead was designed to process hundreds of the same tasks in parallel accepting a higher latency with respect of displaying the result. High performance computing in the range from simple acceleration of an algorithm running on a notebook up to the incorporation in todays super computers heavily influenced both developments over the last years.

The CPUs tend to provide higher memory bandwidth and more parallelism. Here, more parallelism is not restricted to multiple CPU cores. Multiple CPU cores were used in the past to overcome the frequency problem (see Sect. 2.1) and achieve higher performance. More parallelism is also achieved by the trend of wider vector units on CPUs like the Intel AVX technology.

What started for Intel as a "GPU" development, code name Larrabee, is called Xeon Phi since 2012. After several iterations – Knights Ferry and Knights Corner – Intel introduced in late 2012 its first product for super computing to compete with the GPU manufacturers in this field. The addressed trend is clearly shown by the Many Integrated Core (MIC) architecture that provides more than 60 cores with a vector width of 512-bit.

On the GPU side, GPGPU started with a very strict pipeline. This was changed by bringing more flexibility and easier programming with the introduction of unified shaders and CUDA. But still the underlying problems had to fit the highly data parallel pattern in order to achieve high performance. While this flexibility was missing in first CUDA capable device, NVIDIA more and more introduced new technologies

to solve these problems. The trend towards more flexibility is shown with NVIDIA's introduction of a more sophisticated caching system in their Fermi architecture. This is progressed in the Kepler architecture by the introduction of technologies like Dynamic Parallelism and Hyper-Q [Jone 12, NVID 12]. In the long term both technologies – CPUs as well as GPUs – tend to converge towards a flexible and highly parallel processors in order to keep continuing performance improvements.

Giving this general trend the question raises what this means for medical image processing. Therefore, we look at the research on high performance CPU based medical image processing by Hofmann *et al.* In 2009, we showed the general idea how to implement the back-projection operator on Intel's Larrabee [Hofm 09b]. Due to the fact that this acceleration card was never revealed, Hofmann focussed on multi-core CPU systems. The performance results for different applications indicate [Hofm 09a, Hofm 10a, Hofm 10b, Hofm 11] that CPUs follow and try to reduce the gap.

For example Treibig *et al.* showed [Trei 12] an impressive multicore implementation of the FDK approach using the RabbitCT benchmark [Rohk 09]. Utilizing a quad socket system providing 40 CPU cores, the reconstruction time was about 20% faster than those of the four year older GPUs. Even so, a single clear drawback of CPUs as well as the Intel Xeon Phi is still prominent: the lack of texture units. In our approaches the back-projection as well as the forward-projection operator heavily benefit from the GPU provided texture units – analog to Schwarz [Schw 11]. For those algorithms that do not utilize texture units the performance benefits of GPUs compared to the Xeon Phi decrease noticeable and may be seen as comparable.

Answering the above-mentioned question, GPUs will continue to have a significant impact on medical image processing, at least for those algorithms that can utilize texture units – assuming that future GPUs are still equipped with those. For algorithms that cannot take advantage of this computational power, the gap between CPUs and GPUs will close, such that both architectures will be utilized in medical products in the future. In the end it will not be a matter of slightly higher computational performance. The availability in long-term, quality, flexibility, application scope, reusability will be the base arguments. In addition it will be a matter of the provided development environment and tools.

Datasets

Different datasets and acquisition systems were used in the thesis in order to evaluate computational performance as well as image quality aspects. Therefore, the important parameters for these datasets are detailed in this Appendix. Note that for product systems the data was already preprocessed by the manufacturers.

A.1 DataSet A – Human Head

DataSet A was acquired with a short scan acquisition using a C-arm system (Siemens AG, Artis Zee). The dataset consists of 414 projections with an angular increment of 0.5° per projection. In total the C-arm rotated over 207° degree. Each projection provides 1024^2 pixels at an isotropic resolution of 0.32 mm/pixel. The scanned object is a human head. The data was provided with courtesy of the Siemens AG.

A.2 DataSet B – Human Hip

The second dataset referred to as DataSet B was also acquired with a C-arm system. The scanned object is a human hip. The dataset consists of 543 projection also at an isotropic resolution of 0.32 mm/pixel. The acquisition was performed in total over 217° with a projectional increment of 0.4 degree. Each projection provides 1240×960 pixels. The data was provided with courtesy of the Siemens AG.

A.3 DataSet C – Simulated Head

For the iterative algorithms a simulated dataset was utilized and consists of 228 projections. The dataset simulates a short-scan acquisition of a notional C-arm system over 205° with a projectional increment of 0.9 degree. Each projection provides 256×128 pixels at an isotropic resolution of 2.0 mm/pixel. The data was provided with courtesy of Dr. Holger Kunze. For the iterative reconstruction 20 iterations with the following additional parameters were performed:

- Volume: $512 \times 512 \times 350$; isotropic voxel resolution of $0.5 \times 0.5 \times 0.5mm^3$
- Forward-projector sampling step size: $\Delta l = 0.3 \cdot 0.5mm$
- Relaxation factor: $\lambda = 0.3$

A.4 DataSet D – Catphan GTP 528

The fourth dataset was simulated using the Catphan CTP528 phantom that provides 21 line-pairs with different resolutions. DataSet D consists of 400 projections over 360° each providing 1024×128 pixels. These projections provide a resolution of 0.3×0.7 mm/pixel. The data was provided with courtesy of Dr. Michael Balda. For the iterative reconstruction the following additional parameters were used:

- Volume: different resolutions, see Table 5.1
- Forward-projector sampling step size: $\Delta l = 0.3 \cdot \delta x_1$
- Relaxation factor: $\lambda = 0.3$

A.5 DataSet E – Simulated Head

The last angiographic dataset utilizes the head phantom by Dr. G. Lauritsch and Dr. H. Bruder (as available electronically¹). Here, the simulated C-arm acquisition was performed over a limited angle of 180° for a low number of only 90 projections. Each projection provides 1024^2 pixels at an isotropic resolution of 0.4 mm/pixel. For the SART as well as the ML-Convex iterative reconstruction the following additional parameters were used:

- Volume: 512^3 ; isotropic voxel resolution of $0.5 \times 0.5 \times 0.5 \text{mm}^3$
- Forward-projector sampling step size: $\Delta l = 0.3 \cdot 0.5 \text{mm}$
- Relaxation factor: $\lambda = 0.5$

A.6 DataSet F – Female Breasts

Besides the angiographic datasets we utilized a mammography system (Siemens AG, Mammomat Inspiration) for data acquisition. DataSet F represents two datasets acquired over a limited angle of 50° . The 25 projection with a projectional increment of 2° consist of 2816×3584 at an isotropic resolution of only 0.085 mm/pixel. Both scanned objects are a female breast. Differences are only given in the utilized volume resolution. The two datasets were provided with courtesy of the Siemens AG. The following additional parameters were used for the iterative reconstruction:

- Volume: different resolutions, see Section 5.2
- Forward-projector sampling step size: $\Delta l = 0.3 \cdot 0.5 \text{mm}$
- Relaxation factor: $\lambda = 0.3$

¹<http://www.imp.uni-erlangen.de/phantoms>

EM Reconstruction for Computed Tomography

Lange *et al.* states [Lang95] that the reconstruction using the EM algorithm is cumbersome because it entails a large number of exponentiations. Additionally it introduces partial forward-projections for each ray, which multiplies the number of projections to be calculated. This also complicates a parallel implementation of the EM algorithm on GPUs. Therefore this algorithm is less suitable for a high performance implementation and was not considered for the presented research.

In the following we want to summarize the background of the expectation maximization reconstruction for CT. In literature the EM algorithm is well explained and derived from the hidden information principle that states that:

$$\text{observed information} = \text{complete information} - \text{missing information} \quad (\text{B.1})$$

This simple equation is then translated into the statistical framework and specific model. We therefore refer in general to the literature [Demp77, Paul03] for the derivation of the EM algorithm. Next, we summarize the ML-EM algorithm and the suggested approximation by Lange *et al.* for transmission tomography.

B.1 Maximum Likelihood - Expectation Maximization

The EM algorithm as an iterative technique for optimizing the maximum likelihood estimates was first stated [Demp77] in its full generality by Dempster *et al.*

Given an simple experiment – e.g., coin flipping – the complete information consists on the exact coin type and its probability to land on heads [Do08]. For a single coin with unknown probability to land on heads, this can be examined by the maximum likelihood estimation. By repeating the coin flipping experiment a significant amount of times with the same coin and counting the heads, the probability can be estimated respectively determined by the ratio of number of heads divided by total amount of flips.

Expanding this experiment to two or more coins with unknown probability and unknown coin type for each experiment the EM algorithm can be applied. The EM algorithm can be interpreted as a generalization of the maximum likelihood estimation to the incomplete data case [Do 08]. Therefore the EM algorithm attempts to find the parameters – in this case the probabilities of each coin type – that maximize the probability of the observed data meaning the observation of head counts of the different experiments with unknown coin type.

In general the EM algorithm is an iterative method for incomplete data estimation based on two steps. The expectation step and the maximization step. The EM algorithm starts with an initial guess of the parameters. E.g., for the coin example a certain probability to land on heads for each coin. Using this initial parameters a probability distribution over possible completions of each experiment for each coin is computed in the expectation step.

In the maximization step new parameters are then computed using the current completions. Repeating these two steps leads to a convergence of the parameters respectively the missing information. This principle can also be applied to the medical reconstruction case, which is treated next.

B.2 EM Algorithms for Transmission Tomography

In their paper [Lang 84] Lange *et al.* derive the expectation maximization reconstruction algorithm for emission and transmission tomography. In the following the transmission case is summarized and detailed as an extension to Section 3.4.1.

The authors start with describing Poisson distribution and some of its properties. Especially that the collection of independent random variables that follow the Poisson distribution is again a Poisson distribution. For the case of image reconstruction they state that the amount of particles (photons) – respectively the measured energy $d_I(p, r)$ – for the various readings and projections are independent random variables. They argue that this is due to the fact that each reading either occur over disjoint time intervals or involve different angular directions. The likelihood for all readings and all projections reduces to the product of the separate likelihoods for each reading. Therefore the log-likelihood over all readings and all projections reduces to Equation 3.29. In addition they show the strict concavity of the log-likelihood.

Looking at the likelihood for a single reading resp. ray r of a single projection, the expectation step of the EM algorithm is then derived. In particular the set voxels between the source s and the j -th voxel along the measurement for reading r of projection p is defined as $S_{r,j}^{(p)}$, where the j -th voxel is not included.

Given the missing information principle the authors select the numbers of photons entering $\Psi_{r,j}^{(p)}$ each voxel j along the single ray as the complete information. They further describe the key for writing the correct complete data likelihood is to note that the number of photons leaving $\Omega_{r,j}^{(p)}$ each voxel j depends only on the photons entering $\Psi_{r,j}^{(p)}$ and on the product $a_{r,j}^{(p)}\mu_j$ of the corresponding system matrix element and the attenuation coefficient of voxel j .

Further the authors derive that the expectation step requires the conditional expectations $X_{r,j}^{(p)} = E(\Psi_{r,j}^{(p)} | I_0, \boldsymbol{\mu}^{(k)})$ and $Y_{r,j}^{(p)} = E(\Omega_{r,j}^{(p)} | I_0, \boldsymbol{\mu}^{(k)})$ and prove that both conditional expectations can be calculated by:

$$X_{r,j}^{(p)} = d_I(p, r) + I_0 e^{-\sum_{c \in S_{r,j}^{(p)}} a_{r,c}^{(p)} \mu_c^{(k)}} - I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle}, \quad (\text{B.2})$$

$$Y_{r,j}^{(p)} = d_I(p, r) + I_0 e^{-\sum_{c \in S_{r,j}^{(p)} \cup \{j\}} a_{r,c}^{(p)} \mu_c^{(k)}} - I_0 e^{-\langle \mathbf{a}_r^{(p)}, \boldsymbol{\mu}^{(k)} \rangle}. \quad (\text{B.3})$$

Note that the difference for $Y_{r,j}^{(p)}$ to $X_{r,j}^{(p)}$ is that the voxel j is included in $Y_{r,j}^{(p)}$.

For the maximization step Lange *et al.* continue to derive the objective function $Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)})$ of the EM algorithm [Demp 77] out of the partial log-likelihoods, such that:

$$Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)}) = \sum_{p=1}^P \sum_{r=1}^R \sum_{j=1}^J \left[-Y_{r,j}^{(p)} a_{r,j}^{(p)} \mu_j + (X_{r,j}^{(p)} - Y_{r,j}^{(p)}) \cdot \ln \left(1 - e^{-a_{r,j}^{(p)} \mu_j} \right) \right] \quad (\text{B.4})$$

$Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)})$ is then maximized with respect to $\boldsymbol{\mu}$ by taken the partial derivative of $Q(\boldsymbol{\mu} | \boldsymbol{\mu}^{(k)})$ with respect to μ_j and setting it equal to 0. This yields to the transcendental equation

$$\begin{aligned} 0 &= \sum_{p=1}^P \sum_{r=1}^R \left(-Y_{r,j}^{(p)} a_{r,j}^{(p)} \right) + \sum_{p=1}^P \sum_{r=1}^R \left(X_{r,j}^{(p)} - Y_{r,j}^{(p)} \right) \frac{a_{r,j}^{(p)} e^{-a_{r,j}^{(p)} \mu_j}}{1 - e^{-a_{r,j}^{(p)} \mu_j}} \\ &= \sum_{p=1}^P \sum_{r=1}^R \left(-Y_{r,j}^{(p)} a_{r,j}^{(p)} \right) + \sum_{p=1}^P \sum_{r=1}^R \frac{\left(X_{r,j}^{(p)} - Y_{r,j}^{(p)} \right) a_{r,j}^{(p)}}{e^{a_{r,j}^{(p)} \mu_j} - 1}. \end{aligned} \quad (\text{B.5})$$

Due to the fact that this transcendental equation cannot be solved exactly, Lange and Carson suggest an approximation. They argue that all practical application will have sufficiently small voxels so that each term such as $a_{r,j}^{(p)} \mu_j$ will be small. The solution for the unknown μ then can be iteratively approximated - derived from the approximation of the Taylor series - by the following equation (see Eq. 3.30):

$$\mu_j^{(k+1)} = \frac{\sum_{p=1}^P \sum_{r=1}^R (X_{r,j}^{(p)} - Y_{r,j}^{(p)})}{\frac{1}{2} \sum_{p=1}^P \sum_{r=1}^R (X_{r,j}^{(p)} + Y_{r,j}^{(p)}) a_{r,j}^{(p)}}. \quad (\text{B.6})$$

This concludes the summary of the maximum likelihood reconstruction algorithm using the expectation maximization approach for transmission tomography suggested [Lang 84] by Lange and Carson.

List of Figures

1.1	The different generations of CT systems	3
1.2	CT system examples	4
1.3	SPECT/CT and PET/CT systems	5
1.4	C-arm CT systems	6
1.5	Mobile C-arm CT system	7
1.6	Mammography system	8
1.7	Measured signals per second of CT systems over the decades.	9
1.8	Thesis structure graphical overview	18
2.1	Historical CPU and GPU peak computing rates	20
2.2	Historical CPU and GPU memory transfer rates	22
2.3	CPU versus GPU	23
2.4	Graphics pipeline	24
2.5	NVIDIAs G80 chip design	26
2.6	Grid and block decomposition	28
2.7	Instruction planning of a scalar processor	30
2.8	Branching in a streaming multiprocessor	31
2.9	CUDA memory model	32
2.10	NVIDIA's FERMI chip	41
3.1	Geometry description	44
3.2	Detector grid example	45
3.3	Object discretization example	47
3.4	Reconstruction methods overview	51
3.5	Figurative system matrix	58
4.1	Back-projection parallelization scheme	78
4.2	CUDA kernel code for the back-projection	80
4.3	Grid- and block-layout performance difference	81
4.4	Ray casting principle	86
4.5	CUDA kernel code for the forward-projection	89
4.6	CUDA device function code for the ray casting	90
4.7	2-D texture atlas example	92
4.8	CUDA device function code for a manual 3-D texture fetch	93
4.9	Ray casting performance for CUDA 2.0 and OpenGL	96
4.10	FDK reconstruction block diagram	97
4.11	SART reconstruction block diagram using a 2-D texture array	100
4.12	SART reconstruction block diagram using a 3-D texture array	102
4.13	SART reconstruction block diagram using a 2-D texture from pitch-linear memory	103
4.14	ML-Convex 3-D texture block diagram example	108

4.15 ML-Convex 2-D texture block diagram example	109
4.16 OpenCL vs. CUDA performance comparison	111
5.1 Iterative reconstruction of the Catphan CTP528 using SART	114
5.2 Catphan CTP528 SART resolution comparison	115
5.3 Catphan CTP528 SART resolution line-profile comparison	116
5.4 Reconstruction comparison for an angiographic head phantom	117
5.5 3-D mammography reconstructions	118
5.6 3-D mammography resolution comparison	120
5.7 3-D mammography calcification resolution comparison	121
5.8 3-D mammography micro-calc resolution comparison	121
5.9 Log-likelihood plot for the ML-Convex reconstruction	123

List of Tables

2.1	CUDA memory types overview	34
2.2	Utilized graphics cards	35
2.3	OpenCL and CUDA nomenclature	39
3.1	Complexity and convergence of reconstruction algorithms	72
4.1	Pre-/post processing performance example	75
4.2	Back-projection performance example	82
4.3	Bandwidth limitation of the back-projection approach	83
4.4	CUDA grid- and block-size for the forward-projection	94
4.5	Forward projection performance comparison using CUDA and OpenGL	95
4.6	FDK performance comparison	98
4.7	SART CPU-GPU performance comparison	104
4.8	SART GPU implementation comparison	105
4.9	ML-Convex / SART GPU implementation comparison	110
5.1	SART high resolution GPU performance comparison	115
5.2	Performance comparison for limited angle reconstruction	116
5.3	3-D mammography SART reconstruction performance	119
5.4	3-D mammography SART vs. ML-Convex performance comparison .	122

Bibliography

- [Alva 76] R. E. Alvarez and A. Macovski. “Energy-selective Reconstructions in X-ray Computerized Tomography”. *Physics in Medicine and Biology*, Vol. 21, No. 5, pp. 733 – 744, 1976.
- [Ande 84] A. H. Andersen and A. C. Kak. “Simultaneous Algebraic Reconstruction Technique (SART): a superior implementation of the ART algorithm”. *Journal of Ultrasonic Imaging*, Vol. 6, No. 1, pp. 81 – 94, January 1984.
- [Andr 09] A. Andreyev, A. Sitek, and A. Celler. “Acceleration of Blob-Based Iterative Reconstruction Algorithm Using Tesla GPU”. In: B. Yu, Ed., *Nuclear Science Symposium Conference Record, 2009. NSS '09. IEEE*, pp. 4095 – 4098, Orlando, FL, USA, October 2009.
- [Bart 83] C. A. Bartlett. “case ten: EMI and the CT Scanner [A] and [B]”. <http://www.blackwellpublishing.com/grant/docs/10EMI.pdf>, Harvard Business Review Online, June 1983.
- [Bi 09a] W. Bi, Z. Chen, L. Zhang, Y. Xing, and Y. Wang. “A Preliminary Study of OpenCL for Accelerating A Preliminary Study of OpenCL for Accelerating CT Reconstruction and Image Recognition”. In: B. Yu, Ed., *Nuclear Science Symposium Conference Record, 2009. NSS '09. IEEE*, pp. 4059 – 4063, Orlando, FL, USA, October 2009.
- [Bi 09b] W. Bi, Z. Chen, L. Zhang, Y. Xing, and Y. Wang. “Real-Time Visualize the 3D Reconstruction Procedure Using CUDA”. In: B. Yu, Ed., *Nuclear Science Symposium Conference Record, 2009. NSS '09. IEEE*, pp. 883 – 886, Orlando, FL, USA, October 2009.
- [Bock 07] O. Bockenbach, S. Schuberth, M. Knaup, and M. Kachelrieß. “High Performance 3D Image Reconstruction Platforms; State of the Art, Implications and Compromises”. In: *9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, pp. 17 – 20, Lindau, Germany, July 2007.
- [Boyd 90] D. Boyd, S. Ackelsberg, J. Couch, K. Peschmann, and R. Rand. “Historical Perspectives And Recent Progress In The Development Of X-ray Computed Tomography”. In: *Nuclear Science Symposium, 1990. Conference record : Including Sessions on Nuclear Power Systems and Medical Imaging Conference, IEEE*, pp. 1135 – 1141, Crystal City, VA, USA, October 1990.
- [Buhr 09] E. Bührle, B. Keck, S. Böhm, and J. Hornegger. “Mehrstufige zeit- und bewegungsabhängige Rauschreduktion in Echtzeit mittels CUDA”. In: H.-P. Meinzer, T. M. Deserno, H. Handels, and T. Tolxdorff, Eds., *Bildverarbeitung für die Medizin 2009 - Algorithmen Systeme Anwendungen*, pp. 464 – 468, Heidelberg, March 2009.
- [Burt 83] P. J. Burt and E. H. Adelson. “The Laplacian Pyramid as a Compact Image Code”. *Communications, IEEE Transactions on*, Vol. 31, No. 4, pp. 532 – 540, April 1983.

- [Buzu 08] T. M. Buzug. *Computed tomography: from photon statistics to modern cone-beam CT*. Springer Berlin / Heidelberg, 2008.
- [Cent 09a] Centre for Evidence-based Purchasing (CEP). *Buyers' guide: Multi-slice CT scanners*. St George's Healthcare Trust, Medical Physics Department, London, UK, March 2009.
- [Cent 09b] Centre for Evidence-based Purchasing (CEP). *Comparative specifications: 128 to 320 slice CT scanners*. St George's Healthcare Trust, Medical Physics Department, London, UK, March 2009.
- [Cent 09c] Centre for Evidence-based Purchasing (CEP). *Comparative specifications: 64 slice CT scanners*. St George's Healthcare Trust, Medical Physics Department, London, UK, March 2009.
- [Chik 78] É. G. Chikirdin. "Modern methods and equipment for reconstructive tomography (x-ray scanning)". *Biomedical Engineering*, Vol. 12, pp. 314 – 319, November 1978. 10.1007/BF00558614.
- [Corm 63] A. M. Cormack. "Representation of a Function by Its Line Integrals, with Some Radiological Applications". *Journal of Applied Physics*, Vol. 34, No. 9, pp. 2722 – 2727, April 1963.
- [De M 02] B. De Man and S. Basu. "Distance-driven projection and backprojection". In: S. Metzler, Ed., *2002 IEEE Nuclear Science Symposium Conference Record*, pp. 1477 – 1480, Norfolk, Virginia, USA, November 2002.
- [De M 04] B. De Man and S. Basu. "Distance-driven projection and backprojection in three dimensions". *Physics in Medicine and Biology*, Vol. 49, No. 11, pp. 2463 – 2475, June 2004.
- [De M 09] B. De Man and J. Qi. "Statistical Methods for Image Reconstruction". In: *2009 IEEE Nuclear Science Symposium – Short Course Documents*, Orlando, FL, USA, October 2009.
- [De P 93] A. R. De Pierro. "On the relation between the ISRA and the EM algorithm for positron emission tomography". *Medical Imaging, IEEE Transactions on*, Vol. 12, No. 2, pp. 328 – 333, June 1993.
- [De P 95] A. R. De Pierro. "A modified expectation maximization algorithm for penalized likelihood estimation in emission tomography". *Medical Imaging, IEEE Transactions on*, Vol. 14, No. 1, pp. 132 – 137, March 1995.
- [Demp 77] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum Likelihood Reconstruction from Incomplete Data via the EM Algorithm". *Journal of the Royal Statistical Society, Series B (Methodological)*, Vol. 39, No. 1, pp. 1–38, 1977.
- [Denn 05] F. Dennerlein, A. Katsevich, G. Lauritsch, and J. Hornegger. "Exact and efficient cone-beam reconstruction algorithm for a short-scan circle combined with various lines". In: J. M. Fitzpatrick and J. M. Reinhardt, Eds., *Medical Imaging 2005: Physics of Medical Imaging. Proceedings of SPIE*, pp. 388 – 399, February 2005.
- [Denn 08] F. Dennerlein. *Image Reconstruction from Fan-Beam and Cone-Beam Projections*. PhD thesis, Friedrich-Alexander-University Erlangen-Nuremberg, Erlangen, Germany, December 2008.

- [Denn 13] F. Dennerlein and A. Maier. “Approximate truncation robust computed tomography - ATTRACT”. *Physics in Medicine and Biology*, Vol. 58, No. 17, pp. 6133 – 6148, August 2013.
- [Do 08] C. B. Do and S. Batzoglou. “What is the expectation maximization algorithm?”. *Nature Biotechnology*, Vol. 26, No. 8, pp. 897–899, August 2008.
- [Dobb 03] J. T. Dobbins and D. J. Godfrey. “Digital x-ray tomosynthesis: current state of the art and clinical potential”. *Physics in Medicine and Biology*, Vol. 48, No. 19, pp. R65 – R106, September 2003.
- [Doss 00] O. Dössel. *Bildgebende Verfahren in der Medizin: von der Technik zur medizinischen Anwendung*. Springer-Verlag, 2000.
- [Du 12] P. Du, R. Weber, P. Luszczek, S. Tomov, G. Peterson, and J. Dongarra. “From CUDA to OpenCL: Towards a Performance-portable Solution for Multi-platform GPU Programming”. *Journal of Parallel Computing*, Vol. 38, No. 8, pp. 391 – 407, August 2012.
- [Enge 06] K. Engel, M. Hadwiger, J. M. Kniss, C. Rezk-Salama, and D. Weiskopf. *Real-time Volume Graphics*. A. K. Peters, Ltd., Wellesley, Massachusetts, 2006.
- [Enzm 01] D. R. Enzmann, P. M. Anglada, C. Haviley, and L. A. Venta. “Providing Professional Mammography Services: Financial Analysis”. *Radiology*, Vol. 219, No. 1, pp. 467 – 473, May 2001.
- [Fahr 97] R. Fahrig, A. J. Fox, S. Lownie, and D. W. Holdsworth. “Use of a C-arm system to generate true three-dimensional computed rotational angiograms: preliminary in vitro and in vivo results”. *American Journal of Neuroradiology*, Vol. 18, No. 8, pp. 1507 – 1514, September 1997.
- [Feld 84] L. Feldkamp, L. Davis, and J. Kress. “Practical Cone-Beam Algorithm”. *Journal of the Optical Society of America*, Vol. A1, No. 6, pp. 612 – 619, 1984.
- [Fern 03] R. Fernando and M. J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley Longman, Amsterdam, 2003.
- [Fess 08] J. A. Fessler. “Iterative Methods for Image Reconstruction”. In: *5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro – Conference Tutorial*, Paris, France, May 2008.
- [Fill 10] A. G. Filler. “The History, Development and Impact of Computed Imaging in Neurological Diagnosis and Neurosurgery: CT, MRI, and DTI”. *The Internet Journal of Neurosurgery*, Vol. 7, No. 1, 2010.
- [Floh 06] T. Flohr, C. McCollough, H. Bruder, M. Petersilka, K. Gruber, C. Süß, M. Grasruck, K. Stierstorfer, B. Krauss, R. Raupach, A. Primak, A. Küttner, S. Achenbach, C. Becker, A. Kopp, and B. Ohnesorge. “First performance evaluation of a dual-source CT (DSCT) system”. *European Radiology*, Vol. 16, pp. 256 – 268, 2006.
- [Gali 03] R. R. Galigekere, K. Wiesent, and D. W. Holdsworth. “Cone-Beam Reprojection Using Projection-Matrices”. *Medical Imaging, IEEE Transactions on*, Vol. 22, No. 10, pp. 1202 – 1214, October 2003.

- [GE H 11] GE Healthcare. “Veo – CT Model-Based Iterative Reconstruction”. http://www.gehealthcare.com/dose/pdfs/Veo_white_paper.pdf, 2011.
- [Gema 85] S. Geman and D. McClure. “Bayesian Image Analysis: An application to single photon emission tomography”. In: A. S. Association, Ed., *Proceedings of the American Statistical Association. Statistical Computing Section*, pp. 12 – 18, Washington, DC, USA, 1985.
- [Gema 87] S. Geman and D. McClure. “Statistical methods for tomographic image reconstruction”. In: B. I. S. Inst., Ed., *ISI Tokyo session*, pp. 5 – 21, Washington, DC, USA, 1987.
- [Gilb 72] P. Gilbert. “Iterative methods for the three-dimensional reconstruction of an object from projections”. *Journal of Theoretical Biology*, Vol. 36, pp. 105 – 117, 1972.
- [Gord 70] R. Gordon, R. Bender, and G. T. Herman. “Algebraic reconstruction techniques (ART) for three-dimensional electron microscopy and X-ray photography”. *Journal of Theoretical Biology*, Vol. 29, pp. 471 – 481, 1970.
- [Gree 90] P. J. Green. “Bayesian Reconstruction From Emission Tomography Data Using a Modified EM algorithm”. *Medical Imaging, IEEE Transactions on*, Vol. 9, No. 1, pp. 84 – 93, March 1990.
- [Heig 07] B. Heigl and M. Kowarschik. “High-Speed Reconstruction for C-arm Computed Tomography”. In: *9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, pp. 25 – 28, Lindau, Germany, July 2007.
- [Herr 09] J. L. Herraiz, S. España, S. García, R. Cabido, A. S. Montemayor, J. J. V. M. Desco, and J. M. Udias. “GPU Acceleration of a Fully 3D Iterative Reconstruction Software for PET using CUDA”. In: B. Yu, Ed., *Nuclear Science Symposium Conference Record, 2009. NSS '09. IEEE*, pp. 4064 – 4067, Orlando, FL, USA, October 2009.
- [Hill 09a] L. Hillebrand, R. Lapp, Y. Kyriakou, and W. Kalender. “Interactive GPU-accelerated image reconstruction in cone-beam CT”. In: E. Samei and J. Hsieh, Eds., *Proceedings of SPIE*, p. 72582A, SPIE, Orlando, FL, USA, February 2009.
- [Hill 09b] L. Hillebrand. *Echtzeitfähige Rekonstruktion für die Flachdetektor-Computertomographie auf Grafikhardware*. PhD thesis, Institut für Medizinische Physik der Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany, December 2009.
- [Hobe 10] J. Hoberock and D. Tarjan. “Programming Massively Parallel Processors with CUDA – CS193G”. <http://itunes.apple.com/WebObjects/MZStore.woa/wa/viewiTunesUCollection?id=384233322>, iTunesU, Spring 2010.
- [Hofm 09a] H. G. Hofmann, B. Keck, C. Rohkohl, and J. Hornegger. “Putting ‘p’ in RabbitCT - Fast CT Reconstruction Using a Standardized Benchmark”. In: G.-F. PARS, Ed., *Mitteilungen - Gesellschaft für Informatik e.V., Parallel-Algorithmen und Rechnerstrukturen*, pp. 91 – 100, Parsberg, Germany, June 2009.

- [Hofm 09b] H. G. Hofmann, B. Keck, C. Rohkohl, and J. Hornegger. “Towards C-arm CT Reconstruction on Larrabee”. In: B. M. W. Tsui, Ed., *Proceedings of 10th Fully 3D Meeting and 2nd HPIR Workshop*, pp. 1 – 4, Beijing, China, September 2009.
- [Hofm 10a] H. G. Hofmann and J. Hornegger. “Towards real-time C-arm reconstruction using multi-core CPUs”. Intel Corporation, April 2010.
- [Hofm 10b] H. G. Hofmann, B. Keck, and J. Hornegger. “Accelerated C-arm Reconstruction by Out-of-Projection Prediction”. In: T. M. Deserno, H. Handels, H.-P. Meinzer, and T. Tolxdorff, Eds., *Bildverarbeitung für die Medizin 2010*, pp. 380 – 384, Aachen, March 2010.
- [Hofm 11] H. G. Hofmann, B. Keck, C. Rohkohl, and J. Hornegger. “Comparing performance of many-core CPUs and GPUs for static and motion compensated reconstruction of C-arm CT data”. *Medical Physics*, Vol. 38, No. 1, pp. 468 – 473, January 2011.
- [Hopp 08] S. Hoppe, J. Hornegger, G. Lauritsch, F. Dennerlein, and F. Noo. “Truncation Correction for Oblique Filtering Lines”. *Medical Physics*, Vol. 35, No. 12, pp. 5910 – 5920, December 2008.
- [Hsia 10] E. M. Hsiao, F. J. Rybicki, and M. Steigner. “CT coronary angiography: 256-slice and 320-detector row scanners”. *Curr Cardiol Rep*, Vol. 12, No. 1, pp. 68 – 75, January 2010.
- [Huds 94] M. Hudson and R. Larkin. “Accelerated Image Reconstruction using Ordered Subsets of Projection Data”. *Medical Imaging, IEEE Transactions on*, Vol. 13, No. 4, pp. 601 – 609, December 1994.
- [Ino 10] F. Ino, Y. Okitsu, T. Kishia, S. Ohnishi, and K. Hagihara. “Out-of-core cone beam reconstruction using multiple GPUs”. In: *ISBI'10: Proceedings of the 7th IEEE International Symposium on Biomedical Imaging*, pp. 792 – 795, IEEE Press, Rotterdam, Netherlands, April 2010.
- [Inte 05] Intel Corporation. “Excerpts from A Conversation with Gordon Moore: Moore’s Law”. http://large.stanford.edu/courses/2012/ph250/lee1/docs/Excepts_A_Conversation_with_Gordon_Moore.pdf, Online, September 2005.
- [Inte 11] Intel Corporation. “New Levels of CT Image Performance and New Levels in Radiation Dose Management”. <http://www.intel.com/content/dam/doc/white-paper/new-levels-of-ct-radiation-dose-management-paper-.pdf>, Online, 2011.
- [Iser 97] S. Iserhardt. *Fast Cone Beam Reconstruction Using Supported Texture Mapping*. Master’s thesis, Lehrstuhl für Mustererkennung, Institut für Mathematische Maschinen und Datenverarbeitung, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany, May 1997.
- [Jere 10a] A. K. Jerebko, M. Kowarschik, and T. Mertelmeier. “Regularization Parameter Selection in Maximum a Posteriori Iterative Reconstruction for Digital Breast Tomosynthesis”. In: M. Joan, O. Arnau, F. Jordi, and M. Robert, Eds., *Digital Mammography*, pp. 548 – 555, Springer Berlin / Heidelberg, June 2010.

- [Jere 10b] A. K. Jerebko and T. Mertelmeier. “Evaluation and optimization of the maximum-likelihood approach for image reconstruction in digital breast tomosynthesis”. In: E. Samei and N. J. Pelc, Eds., *Medical Imaging 2010: Physics of Medical Imaging, Proceedings of SPIE*, p. 76220E, SPIE, San Diego, CA, USA, February 2010.
- [Jone 12] S. Jones. “Inside the Kepler Architecture”. In: *GPU Technology Conference 2012*, p. SB032, San Jose, CA, USA, May 2012.
- [Jose 82] P. M. Joseph. “An Improved Algorithm for Reprojecting Rays Through Pixel Images”. *Medical Imaging, IEEE Transactions on*, Vol. MI-1, No. 3, pp. 192 – 196, November 1982.
- [Kach 07] M. Kachelrieß, M. Knaup, and O. Bockenbach. “Hyperfast parallel-beam and cone-beam backprojection using the cell general purpose hardware”. *Medical Physics*, Vol. 34, No. 4, pp. 1474 – 1486, March 2007.
- [Kacz 37] S. Kaczmarz. “Angenäherte Auflösung von Systemen linearer Gleichungen”. *Bulletin International de l’Academie Polonaise des Sciences et des Lettres*, Vol. 6-8A, pp. 335 – 357, 1937.
- [Kak 88] A. C. Kak and M. Slaney. *Principles of Computerized Tomographic Imaging*. IEEE Press, 1988.
- [Kale 90] W. A. Kalender, W. Seissler, E. Klotz, and P. Vock. “Spiral volumetric CT with single-breath-hold technique, continuous transport, and continuous scanner rotation”. *Radiology*, Vol. 176, No. 1, pp. 181 – 183, July 1990.
- [Kant 08] D. Kanter. “NVIDIA’s GT200: Inside a Parallel Processor”. <http://www.realworldtech.com/gt200/>, Real World Technologies, August 2008.
- [Kant 09] D. Kanter. “Inside Fermi: Nvidia’s HPC Push”. <http://www.realworldtech.com/fermi/>, Real World Technologies, September 2009.
- [Kats 02] A. Katsevich. “Theoretically Exact Filtered Backprojection-Type Inversion Algorithm for Spiral CT”. *SIAM Journal of Applied Mathematics*, Vol. 62, No. 6, pp. 2012 – 2026, July 2002.
- [Keck 09a] B. Keck, H. G. Hofmann, H. Scherl, M. Kowarschik, and J. Hornegger. “GPU-accelerated SART reconstruction using the CUDA programming environment”. In: E. Samei and J. Hsieh, Eds., *Proceedings of SPIE*, p. 72582B, Orlando, FL, USA, February 2009.
- [Keck 09b] B. Keck, H. G. Hofmann, H. Scherl, M. Kowarschik, and J. Hornegger. “High Resolution Iterative CT Reconstruction using Graphics Hardware”. In: B. Yu, Ed., *2009 IEEE Nuclear Science Symposium Conference Record*, pp. 4035 – 4040, Orlando, FL, USA, October 2009.
- [Keck 10] B. Keck, M. Kowarschik, J. Ludwig, T. Mertelmeier, and H. Scherl. “Reduction of artifacts caused by movement of an X-ray tube in object reconstruction”. Patent application, February 2010.
- [Khro 10] Khronos Group. “OpenCL API 1.1 Quick Reference Card”. <http://www.khronos.org/opencl/>, 2010.

- [Kirk 10] D. B. Kirk and W. W. Hwu. *Programming Massively Parallel Processors - A Hands-on Approach*. Morgan Kaufmann - Elsevier, 30 Corporate Drive, Suite 400, Burlington, MA 01803, USA, 2010.
- [Knau 08] M. Knaup, S. Steckmann, and M. Kachelriess. "GPU-based parallel-beam and cone-beam forward- and backprojection using CUDA". In: *Nuclear Science Symposium Conference Record, 2008. NSS '08. IEEE*, pp. 5153 – 5157, Dresden, Germany, October 2008.
- [Kram 07] R. Kramme, Ed. *Medizintechnik: Verfahren - Systeme - Informationsverarbeitung*. Springer Medizin Verlag Heidelberg, 2007.
- [Kran 99] S. G. Krantz. *Handbook of Complex Variables*, Chap. §9.1.3, p. 118. Birkhäuser Boston, 1999.
- [Kunz 07] H. Kunze. *Iterative Rekonstruktion in der Medizinischen Bildverarbeitung*. PhD thesis, Friedrich-Alexander-University Erlangen-Nuremberg, Erlangen, Germany, June 2007.
- [Lang 84] K. Lange and R. Carson. "EM reconstruction algorithms for emission and transmission tomography". *Journal of Computer Assisted Tomography*, Vol. 8, No. 2, pp. 306 – 316, April 1984.
- [Lang 87] K. Lange, M. Bahn, and R. Little. "A theoretical study of some maximum likelihood algorithms for emission and transmission tomography". *Medical Imaging, IEEE Transactions on*, Vol. 6, No. 2, pp. 106 – 114, June 1987.
- [Lang 95] K. Lange and J. A. Fessler. "Globally convergent algorithms for maximum a posteriori transmission tomography". *Image Processing, IEEE Transactions on*, Vol. 4, No. 10, pp. 1430 – 1438, October 1995.
- [Lee 10] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupati, P. Hammarlund, R. Singhal, and P. Dubey. "Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU". *SIGARCH Comput. Archit. News*, Vol. 38, No. 3, pp. 451 – 460, June 2010.
- [Lu 09] Y. Lu, W. Wang, S. Chen, Y. Xie, J. Qin, W.-M. Pang, and P.-A. Heng. "Accelerating Algebraic Reconstruction Using CUDA-Enabled GPU". In: *Computer Graphics, Imaging and Visualization, International Conference on*, pp. 480 – 485, IEEE Computer Society, Los Alamitos, CA, USA, 2009.
- [Lueb 08] D. Luebke. "CUDA: SCALABLE PARALLEL PROGRAMMING FOR HIGH-PERFORMANCE SCIENTIFIC COMPUTING". In: *ISBI'08: Proceedings of the Fifth IEEE international Symposium on Biomedical Imaging*, pp. 836 – 838, IEEE Press, 2008.
- [Maco 83] A. Macovski. *Medical Imaging Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ 07458, March 1983.
- [Manh 12] M. Manhart, M. Kowarschik, A. Fieselmann, Y. Deuerling-Zheng, and J. Hornegger. "Fast Dynamic Reconstruction Algorithm with Joint Bilateral Filtering for Perfusion C-arm CT". In: B. Yu, Ed., *Nuclear Science Symposium Conference Record, 2012. NSS '12. IEEE*, pp. 2304 – 2311, Anaheim, CA, USA, October 2012.

- [Manh 13] M. Manhart, M. Kowarschik, A. Fieselmann, Y. Deuerling-Zheng, K. Royalty, A. Maier, and J. Hornegger. “Dynamic Iterative Reconstruction for Interventional 4-D C-Arm CT Perfusion Imaging”. *Medical Imaging, IEEE Transactions on*, Vol. 32, No. 7, pp. 1336 – 1348, April 2013.
- [Muel 00] K. Mueller and R. Yagel. “Rapid 3-D cone-beam reconstruction with the simultaneous algebraic reconstruction technique (SART) using 2-D texture mapping hardware”. *Medical Imaging, IEEE Transactions on*, Vol. 19, No. 12, pp. 1227 – 1237, December 2000.
- [Muel 07] K. Mueller, F. Xu, and N. Neophytou. “Why do Commodity Graphics Hardware Boards (GPUs) work so well for acceleration of Computed Tomography?”. In: *SPIE Electronic Imaging Conference*, San Jose, CA, USA, January 2007. (Keynote, Computational Imaging V).
- [Muel 98a] K. Mueller. *Fast and accurate three-dimensional reconstruction from cone-beam projection data using algebraic methods*. PhD thesis, School of The Ohio State University, Columbus, OH, USA, 1998.
- [Muel 98b] K. Mueller and R. Yagel. “Rapid 3D cone-beam reconstruction with the Algebraic Reconstruction Technique (ART) by utilizing texture mapping graphics hardware”. In: *Proceedings of the Nuclear Science Symposium*, pp. 1552 – 1559, Toronto, Canada, November 1998.
- [Muel 99] K. Mueller and R. Yagel. “Use of Graphics Hardware to Accelerate Algebraic Reconstruction Methods”. In: J. M. Boone and J. T. Dobbins, Eds., *Proceedings of SPIE*, pp. 615 – 625, SPIE, San Diego, CA, USA, May 1999.
- [Natt 86] F. Natterer. *The Mathematics of Computerized Tomography*. Wiley, 1986.
- [Nguy 09] V.-G. Nguyen, S.-J. Lee, and M. N. Lee. “GPU Accelerated Statistical Image Reconstruction for Compton Cameras”. In: B. Yu, Ed., *Nuclear Science Symposium Conference Record, 2009. NSS '09. IEEE*, pp. 3550 – 3555, Orlando, FL, USA, October 2009.
- [Nico 08] S. Nicolaou. “The Use of the Dual Source CT in the Emergency Polytrauma Patient”. In: *10th Annual International Symposium on Multidetector-Row CT*, Wynn, Las Vegas, Nevada, May 2008.
- [NVID 07a] NVIDIA Corp. “NVIDIA Compute Unified Device Architecture - C Programming Guide”. <http://www.nvidia.com/cuda>, 2007.
- [NVID 07b] NVIDIA Corp. “NVIDIA Compute Unified Device Architecture - CUFFT Library”. <http://www.nvidia.com/cuda>, October 2007.
- [NVID 07c] NVIDIA Corp. “NVIDIA Compute Unified Device Architecture - Reference Manual”. <http://www.nvidia.com/cuda>, 2007.
- [NVID 10a] NVIDIA Corp. “NVIDIA CUDA - CUDA C Best Practices Guide”. <http://www.nvidia.com/cuda>, 2010.
- [NVID 10b] NVIDIA Corp. “NVIDIA’s Next Generation CUDA Compute Architecture: Fermi”. <http://www.nvidia.com/fermi>, January 2010.
- [NVID 10c] NVIDIA Corp. “OpenCL - Programming Guide for the CUDA Architecture”. <http://www.nvidia.com/opencl>, February 2010.

- [NVID 12] NVIDIA Corp. “NVIDIA’s Next Generation CUDATM Compute Architecture: NVIDIA’s Next Generation CUDA Compute Architecture: Kepler GK110”. <http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf>, May 2012.
- [Owen 05] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. “A Survey of General-Purpose Computation on Graphics Hardware”. In: *In Eurographics 2005 - State of the Art Report*, pp. 21 – 51, Eurographics Association, Konstanz, Germany, June 2005.
- [Pan 09] Y. Pan, R. Whitaker, A. Cheryauka, and D. Ferguson. “Feasibility of GPU-assisted iterative image reconstruction for mobile C-arm CT”. In: E. Samei and J. Hsieh, Eds., *Proceedings of SPIE*, p. 72585J, SPIE, Orlando, FL, USA, February 2009.
- [Pan 10] Y. Pan, R. Whitaker, A. Cheryauka, and D. Ferguson. “TV-regularized Iterative Image Reconstruction on a Mobile C-ARM CT”. In: E. Samei and N. J. Pelc, Eds., *Proceedings of SPIE*, p. 76222L, SPIE, San Diego, CA, USA, February 2010.
- [Park 82] D. L. Parker. “Technical Note: Optimal short scan convolution reconstruction for fan beam CT”. *Medical Physics*, Vol. 9, No. 2, pp. 254 – 257, March 1982.
- [Paul 03] D. W. R. Paulus and J. Hornegger. *Applied Pattern Recognition*. Vieweg, 4th edition Ed., February 2003.
- [Phar 05] M. Pharr. *GPU Gems 2: Techniques for Graphics and Compute-Intensive Programming*. Addison-Wesley Longman, March 2005.
- [Pola 07] A. Polacin. “Design Consideration on Image Reconstruction System for High-End CT Scanner”. In: *9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, pp. 21 – 24, Lindau, Germany, July 2007.
- [Prat 09] G. Pratz, G. Chin, P. D. Olcott, and C. S. Levin. “Fast, Accurate and Shift-Varying Line Projections for Iterative Reconstruction Using the GPU”. *Medical Imaging, IEEE Transactions on*, Vol. 28, No. 3, pp. 435 – 445, March 2009.
- [Rado 17] J. Radon. “Über die Bestimmung von Funktionen durch ihre Integralwerte laengs gewisser Mannigfaltigkeit”. *Berichte Saechs. Akad. Wiss., Math. Phys. Kl.*, Vol. 69, pp. 262 – 277, 1917.
- [Riab 07] D. Riabkov, X. Xue, D. Tubbs, and A. Cheryauka. “Accelerated cone-beam backprojection using GPU-CPU hardware”. In: *9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, pp. 68 – 71, Lindau, Germany, July 2007.
- [RKI 12] RKI. “Krebs in Deutschland 2007/2008”. Tech. Rep. 8. Ausgabe, Robert Koch-Institut (Hrsg) and Gesellschaft der epidemiologischen Krebsregister in Deutschland e.V. (Hrsg), Berlin, 2012.
- [Rohk 09] C. Rohkohl, B. Keck, H. G. Hofmann, and J. Hornegger. “Technical Note: RabbitCT – an open platform for benchmarking 3D cone-beam reconstruction algorithms”. *Medical Physics*, Vol. 36, No. 9, pp. 3940 – 3944, September 2009.

- [Sche 07a] H. Scherl, S. Hoppe, F. Dennerlein, G. Lauritsch, W. Eckert, M. Kowarschik, and J. Hornegger. “On-the-fly-Reconstruction in Exact Cone-Beam CT using the Cell Broadband Engine Architecture”. In: *Proceedings Fully3D Meeting and HPIR Workshop*, pp. 29 – 32, Lindau, Germany, July 2007.
- [Sche 07b] H. Scherl, B. Keck, M. Kowarschik, and J. Hornegger. “Fast GPU-Based CT Reconstruction using the Common Unified Device Architecture (CUDA)”. In: *Nuclear Science Symposium Conference Record, 2007. NSS '07. IEEE*, pp. 4464 – 4466, October 2007.
- [Sche 07c] H. Scherl, M. Koerner, H. G. Hofmann, W. Eckert, M. Kowarschik, and J. Hornegger. “Implementation of the FDK Algorithm for Cone-Beam CT on the Cell Broadband Engine Architecture”. In: J. Hsieh and M. J. Flynn, Eds., *Medical Imaging 2007: Physics of Medical Imaging. Proceedings of SPIE*, p. 651058, March 2007.
- [Sche 10] H. Scherl. “High-Speed CT Reconstruction in Medical Diagnosis & Industrial NDT Applications”. In: *GPU Technology Conference 2010*, p. 2096, San Jose, CA, USA, May 2010.
- [Sche 11] H. Scherl. *Evaluation of State-of-the-Art Hardware Architectures for Fast Cone-Beam CT Reconstructions*. Vieweg+Teubner, 2011.
- [Sche 12] H. Scherl, M. Kowarschik, H. G. Hofmann, B. Keck, and J. Hornegger. “Evaluation of State-of-the-Art Hardware Architectures for Fast Cone-Beam CT Reconstruction”. *Journal of Parallel Computing*, Vol. 38, No. 3, pp. 111 – 124, March 2012.
- [Schi 06] T. Schiwietz, T. Chang, P. Speier, and R. Westermann. “MR image reconstruction using the GPU”. In: M. J. Flynn and J. Hsieh, Eds., *Proceedings of SPIE*, p. 61423T, SPIE, 2006.
- [Schm 04] P. Schmitt. “A Brief History of Computed Tomography”. http://www.medical.siemens.com/siemens/en_GB/rg_marcom_FBAs/files/brochures/magazin2_2004/P8-9_CoverStory_CT-History.pdf, Siemens Medical Solutions, October 2004.
- [Schw 11] K. Schwarz. “The Texture Unit as Performance Booster in CT-Reconstruction”. In: *NVIDIA Tesla GPU Computing at the International Supercomputing Conference 2011*, Hamburg, Germany, June 2011.
- [Schw 13] C. Schwemmer, C. Rohkohl, G. Lauritsch, K. Müller, and J. Hornegger. “Residual Motion Compensation in ECG-Gated Interventional Cardiac Vasculature Reconstruction”. *Physics in Medicine and Biology*, Vol. 58, No. 11, pp. 3717 – 3737, May 2013.
- [Shep 74] L. A. Shepp and B. Logan. “Reconstructing Interior Head Tissue From X-ray Transmissions”. *Nuclear Science, IEEE Transactions on*, Vol. 21, No. 1, pp. 228 – 236, February 1974.
- [Shep 82] L. A. Shepp and Y. Vardi. “Maximum Likelihood Reconstruction for Emission Tomography”. *Medical Imaging, IEEE Transactions on*, Vol. 1, No. 2, pp. 113 – 122, October 1982.
- [Shir 02] P. Shirley. *Fundamentals of Computer Graphics*. A. K. Peters, Ltd., Natick, MA, USA, 2002.

- [Sidd 85] R. L. Siddon. “Fast calculation of the exact radiological path for a three-dimensional CT array”. *Medical Physics*, Vol. 12, No. 2, pp. 252 – 255, April 1985.
- [Sieg 11a] C. Siegl, H. G. Hofmann, B. Keck, M. Prümmer, and J. Hornegger. “Hardware-unabhängige Beschleunigung von Medizinischer Bildverarbeitung mit OpenCL”. In: H. Handels, J. Ehrhardt, T. M. Deserno, H.-P. Meinzer, and T. Tolxdorff, Eds., *Bildverarbeitung für die Medizin 2011*, pp. 449 – 453, Springer Berlin Heidelberg 2011, Berlin, March 2011.
- [Sieg 11b] C. Siegl, H. G. Hofmann, B. Keck, M. Prümmer, and J. Hornegger. “OpenCL, a Viable Solution for High-performance Medical Image Reconstruction?”. In: N. J. Pelc, E. Samei, and R. M. Nishikawa, Eds., *Proceedings of SPIE*, p. 79612Q, Orlando, FL, USA, February 2011.
- [Siem 04] Siemens AG. *Allgemeine Informationen zur Mammographie und Brustvorsorge*. Printed, Siemens AG, Medical Solutions, Spezialarbeitsplätze, Allee am Röthelheimpark 2, D-91052 Erlangen, Germany, 2004.
- [Stie 00] K. Stierstorfer. “Drasim: A CT-simulation tool”. Internal Report, Siemens Healthcare, 2000.
- [Stie 04] K. Stierstorfer, A. Rauscher, J. Boese, H. Bruder, S. Schaller, and T. Flohr. “Weighted FBP — a simple approximate 3D FBP algorithm for multislice spiral CT with good dose usage for arbitrary pitch”. *Physics in Medicine and Biology*, Vol. 49, No. 11, pp. 2209 – 2218, 2004.
- [Stro 03] N. Strobel, B. Heigl, T. Brunner, O. Schütz, M. Mitschke, K. Wiesent, and T. Mertelmeier. “Improving 3D Image Quality of X-ray C-Arm Imaging Systems by Using Properly Designed Pose Determination Systems for Calibrating the Projection Geometry”. In: M. J. Yaffe and L. E. Antonuk, Eds., *Medical Imaging 2007: Physics of Medical Imaging. Proceedings of SPIE*, pp. 943 – 954, 2003.
- [Stro 09] N. Strobel, O. Meissner, J. Boese, T. Brunner, B. Heigl, M. Hoheisel, G. Lauritsch, M. Nagel, M. Pfister, E.-P. Rührnschopf, B. Scholz, B. Schreiber, M. Spahn, M. Zellerhoff, and K. Klingenberg-Regn. *Multislice CT*, Chap. 3D Imaging with Flat-Detector C-Arm Systems, pp. 33 – 51. Springer Berlin / Heidelberg, 3 Ed., 2009.
- [Sunn 09] J. Sunnegårdh. *Iterative Filtered Backprojection Methods for Helical Cone-Beam CT*. PhD thesis, Dept. Elect. Eng., Linköping University, Sweden, September 2009.
- [Trei 12] J. Treibig, G. Hager, H. G. Hofmann, J. Hornegger, and G. Wellein. “Pushing the limits for medical image reconstruction on recent standard multicore processors”. *International Journal of High Performance Computing Applications*, p. 13, June 2012.
- [Turb 01] H. Turbell. *Cone-Beam Reconstruction Using Filtered Backprojection*. PhD thesis, Dept. Elect. Eng., Linköping University, Sweden, February 2001.
- [Vaz 07] M. S. Vaz, M. McLin, and A. Ricker. “Current and Next Generation GPUs for Accelerating CT Reconstruction: Quality, Performance, and Tuning”. In: *9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, pp. 76 – 79, Lindau, Germany, July 2007.

- [Vint 10] D. Vintache, B. Humbert, and D. Brasse. “Iterative Reconstruction for Transmission Tomography on GPU Using Nvidia CUDA”. *Tsinghua Science and Technology*, Vol. 15, No. 1, pp. 11 – 16, 2010.
- [Volk 08a] V. Volkov and J. W. Demmel. “Benchmarking GPUs to tune dense linear algebra”. In: *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pp. 1 – 11, IEEE Press, Piscataway, NJ, USA, 2008.
- [Volk 08b] V. Volkov and B. Kazian. “Fitting FFT onto the G80 Architecture”. http://www.cs.berkeley.edu/~kubitron/courses/cs258-S08/projects/reports/project6_report.pdf, Online, May 2008.
- [Volk 10] V. Volkov. “Better Performance at Lower Occupancy”. <http://www.cs.berkeley.edu/~volkov/volkov10-GTC.pdf>, Online, May 2010.
- [Wang 09] Y. Wang, T. Feng, L. Shen, and Y. Xing. “Research on ATI-CAL for Accelerating FBP Reconstruction”. In: B. Yu, Ed., *Nuclear Science Symposium Conference Record, 2009. NSS '09. IEEE*, pp. 4126 – 4129, Orlando, FL, USA, October 2009.
- [Wang 10] B. Wang, L. Zhu, K. Jia, and J. Zheng. “Accelerated cone beam CT reconstruction based on OpenCL”. In: *Image Analysis and Signal Processing (IASP), 2010 International Conference on*, pp. 291 – 295, Zhejiang, China, April 2010.
- [Wein 08] A. Weinlich, B. Keck, H. Scherl, M. Kowarschik, and J. Hornegger. “Comparison of High-Speed Ray Casting on GPU using CUDA and OpenGL”. In: R. Buchty and J.-P. Weiß, Eds., *Proceedings of the First International Workshop on New Frontiers in High-performance and Hardware-aware Computing (HipHaC'08)*, pp. 25 – 30, Lake Como, Italy, November 2008.
- [Weis] E. W. Weisstein. “Jensen’s Inequality”. <http://mathworld.wolfram.com/JensensInequality.html>, From MathWorld – A Wolfram Web Resource.
- [Wiec 04] A. Wiecek. “Siemens Medical Solutions Celebrates 30 Years of Computed Tomography”. Siemens Press, June 2004.
- [Wies 00] K. Wiesent, K. Barth, N. Navab, P. Durlak, T. Brunner, O. Schütz, and W. Seissler. “Enhanced 3-D-reconstruction algorithm for C-arm systems suitable for interventional procedures.”. *Medical Imaging, IEEE Transactions on*, Vol. 19, No. 5, pp. 391 – 403, May 2000.
- [Xu 04] F. Xu and K. Mueller. “Ultra-fast 3D filtered backprojection on commodity graphics hardware”. In: *Proceedings of the International Symposium on Biomedical Imaging*, pp. 571 – 574, Arlington, VA, USA, April 2004.
- [Xu 05] F. Xu and K. Mueller. “Accelerating Popular Tomographic Reconstruction Algorithms on Commodity PC Graphics Hardware”. *Nuclear Science, IEEE Transactions on*, Vol. 52, No. 3, pp. 654 – 663, 2005.
- [Xu 06] F. Xu and K. Mueller. “A comparative study of popular interpolation and integration methods for use in computed tomography”. In: *3rd IEEE International Symposium on Biomedical Imaging: Nano to Macro*, pp. 1252 – 1255, IEEE, Arlington, VA, USA, April 2006.

- [Xu 07a] F. Xu and K. Mueller. “Real-time 3D computed tomographic reconstruction using commodity graphics hardware”. *Physics in Medicine and Biology*, Vol. 52, pp. 3405 – 3419, July 2007.
- [Xu 07b] F. Xu and K. Mueller. “GPU-Acceleration of Attenuation and Scattering Compensation in Emission Computed Tomography”. In: *Proceedings Fully3D Meeting and HPIR Workshop*, pp. 33 – 36, Lindau, Germany, July 2007.
- [Xu 09a] F. Xu, A. Khamene, and O. Fluck. “High performance tomosynthesis enabled via a GPU-based iterative reconstruction framework”. In: E. Samei and J. Hsieh, Eds., *Proceedings of SPIE*, p. 72585A, Orlando, FL, USA, February 2009.
- [Xu 09b] W. Xu and K. Mueller. “Accelerating regularized iterative CT reconstruction on commodity graphics hardware (GPU)”. In: *ISBI'09: Proceedings of the Sixth IEEE international Symposium on Biomedical Imaging*, pp. 1287 – 1290, IEEE Press, Piscataway, NJ, USA, 2009.
- [Xu 10] F. Xu, W. Xu, M. Jones, B. Keszthelyi, J. Sedat, D. Agard, and K. Mueller. “On the efficiency of iterative ordered subset reconstruction algorithms for acceleration on GPUs”. *Comput. Methods Prog. Biomed.*, Vol. 98, No. 3, pp. 261 – 270, 2010.
- [Yang 07] H. Yang, M. Li, K. Koizumi, and H. Kudo. “Accelerating Backprojections via CUDA Architecture”. In: *Proceedings Fully3D Meeting and HPIR Workshop*, pp. 52 – 55, Lindau, Germany, July 2007.
- [Zeng 00] G. L. Zeng and G. T. Gullberg. “Unmatched projector/backprojector Pairs in an Iterative Reconstruction Algorithm”. *IEEE Transactions on Medical Imaging*, Vol. 19, No. 5, pp. 548 – 555, May 2000.
- [Zeng 10] G. L. Zeng. *Medical Image Reconstruction: A Conceptual Tutorial*. Springer New York, 2010.